UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

ING. TÉCNICA INDUSTRIAL

(ESPECIALIDAD MECÁNICA)

PROYECTO FIN DE CARRERA

APLICACIÓN DEL PROCESO DE INGENIERÍA PARA EL DESARROLLO DE UN SOFTWARE DE APRENDIZAJE

Alumno: D. Antonio Castells de Castro

Tutor: D. José María de Fuentes García Romero de Tejada

Co-Tutora: Dña. Lorena González Manzano

Leganés, a 13 de septiembre de 2012

AGRADECIMIENTOS

Todos sabemos lo gratificante que es verse reflejado en los agradecimientos de cualquier documento. Por ello, esperemos no olvidar a nadie y si es así, deciros que lo importante es la ayuda.

En primer lugar quiero agradecer y dedicar este proyecto a mis padres, cuyo esfuerzo día a día me han dado motivación y madurez para acabar esta carrera. Os lo merecéis tanto o más que yo.

A mis tutores, Lorena y José María, cuya profesionalidad y entrega a lo largo de todo el proyecto han sido más que necesarias. Gracias por motivarme y sobre todo por confiar en que podría llevar a cabo un proyecto de informática.

A José Miguel Martínez, más conocido como Fito, cuya profesionalidad en todo lo que respecta al Bridge ha sido de gran ayuda. Gracias por estar ahí en todo lo que he necesitado. A su vez, agradecer a Alejandro Valdivieso el diseño de las cartas.

A mis tíos y a mi padre por haberme enseñado este maravilloso juego. Además, gracias por darme la oportunidad de trabajar codo con codo aprendiendo cada día más.

A mis amigos, tanto universitarios como los de toda la vida. Gracias por el interés y el apoyo mostrados en esos momentos de reunión.

Al Bridge como disciplina deportiva y de entretenimiento. Espero no desvincularme nunca de él. También agradecer a los jugadores del club Squeeze porque ellos también han aportado su granito de arena.

Agradecer también a la Universidad Carlos III esta etapa universitaria que sin duda ha sido y será inolvidable.

Y por último, a ti lector, por el interés mostrado leyendo este documento. Si no conoces el Bridge, espero que este proyecto te motive a iniciarte.

ÍNDICE

1. INTRODUCCIÓN 14
1.1 Proceso de ingeniería 14
1.2 Motivación. 15
1.3 Introducción histórica al Bridge 15
1.4 Tecnicismos utilizados 17
1.5 Objetivo del proyecto 19
1.6 Alcance del Proyecto 19
1.7 Organización del presente documento 20
2. ANÁLISIS 21
2.1 Estado del arte 21
2.1.1 Learn To Play Bridge 21
2.1.2 Bridge Base Online (B.B.O) 22
2.1.3 BBO IOS & Android application 23
2.1.4 Bridge Master 2000 23
2.1.5 Jack Bridge 24
2.1.6 Blue Chip MiniBridge 25
2.1.7 Bridge Baron 22 26
2.1.8 Q-Plus Bridge 26
2.1.9 Fun Bridge 27
2.1.10 Micro MiniBridge 12 28
2.1.11 GIB 28
2.2 Comparativa de los programas presentados 29
2.3 Análisis de tecnologías 35
2.3.1 Lenguaje de programación C++ 35
2.3.2 Lenguaje de programación Java 35
2.3.3 Lenguaje de programación JavaScript 36
2.3.4 Lenguaje de programación Groovy 36
2.4 Dealmaster Pro 36
2.5 Fichero TXT 37
2.6 Diagramas de casos de uso. 38
2.7 Catálogo de requisitos del proyecto. 43

2.7.1 Esquema de la tabla de requisitos 43
2.7.2 Requisitos Funcionales 44
2.7.3 Requisitos no Funcionales 48
2.7.4 Requisitos inversos 50
2.8 Arquitectura preliminar 50
2.9 Plan de pruebas del software 51
2.9.1 Formato de tabla del plan de pruebas 52
2.9.2 Pruebas del software 52
2.10 Análisis del juego 58
2.10.1 Técnicas de programación aplicables. 58
2.10.2 Lógica de juego del carteo 58
2.10.3 Lógica de juego de la subasta 61
2.10.4 Puntuación 62
3. DISEÑO 64
3.1 Selección de tecnologías 64
3.1.1 Lenguaje de programación 64
3.1.2 Entorno de programación. 65
3.1.3 Programación de la lógica de juego. 65
3.1.4 Diseño de los colores de la pantalla de las partidas 65
3.1.5 Diseño de las imágenes de los tutoriales. 65
3.2 Diagrama de componentes y de despliegue 66
3.3 Diseño de los componentes 67
3.3.1 Diseño de la interfaz del componente Vista 67
3.3.2 Diagrama de clases del componente Vista 69
3.3.3 Diagrama de clases del subcomponente *VistaPartida* 70
3.3.4 Diagramas de clases del subcomponente Controlador Juego 71
3.3.5 Diagrama de clases de los subcomponentes JuegoBridge y Partida 74
3.3.6 Componente Modelo 76
3.3.7 Diseño de los iconos las clases Carta y Voz. 77
3.4 Diagramas de Secuencia 78
3.4.1 Diagrama del caso de uso CU.01: Mostrar tutoriales de Bridge 79
3.4.2 Diagrama del caso de uso CU.02: Jugar partida MiniBridge 79
3.4.3 Diagrama del caso de uso CU.03: Jugar partida Bridge. 84

4. IMPLEMENTACIÓN DEL SOFTWARE 89
4.1 Proceso de codificación del componente Vista 89
4.1.1 Subcomponente Vista Partida 89
4.1.2 Subcomponente VistaTutorial 97
4.1.3 Subcomponente VistaMenu 99
4.1.4 Clase PantallaBienvenida 101
4.2 Proceso de codificación del componente Controlador 102
4.2.1 Controlador Juego 102
4.2.2 Juego Bridge y partida 105
4.2.3 Jugador 105
4.3 Proceso de Codificación del componente Modelo 106
4.4 Proceso de rediseño de las imágenes 108
4.5 Resultados de las pruebas de aceptación 109
5. CONCLUSIONES Y LÍNEAS FUTURAS 110
5.1 Conclusiones finales 110
5.2 Dificultad del proyecto 111
5.3 Líneas futuras 113
BIBLIOGRAFÍA Y REFERENCIAS 115
ANEXO I. GESTIÓN DEL PROYECTO 117
Modelo de proceso elegido 117
Planificación del trabajo 118
Planificación inicial 118
Desarrollo real 119
Análisis de la desviación 119
Medios técnicos Empleados 121
Análisis económico del proyecto 122
Análisis de costes 122
Costes iniciales 125
Costes finales 126
ANEXO II. MANUAL DE USUARIO 127
Menú de juego 127
Tutoriales 128
Partida de miniBridge 128

Partida de Bridge 129
ANEXO III. INTRODUCCIÓN A LA PARTIDA DE BRIDGE 130
Introducción y objetivos 130
Presentación de la baraja 130
Fases del Juego 131
Bidding Box 132
Subasta 133
Sistema de la Subasta 134
Reglas de la subasta 134
Carteo 138
Sistema del Carteo 138
Reglas del Carteo 139
Tipos de partidas 143
Partida MiniBridge 144
Partida de Bridge Duplicado 144
Partida libre, Rubber 145
Partida por equipos 145

ÍNDICE DE TABLAS

Tabla 1. Formato de tabla de propiedades de los programas de bridge estudiados 32
Tabla 2. Tabla de propiedades de los programas de Bridge estudiados 33
Tabla 3. Formato de la descripción de casos de uso 40
Tabla 4. Caso de uso CU.01 40
Tabla 5. Caso de uso CU.02 41
Tabla 6. Caso de uso CU.03 42
Tabla 7. Formato de especificación de requisitos 43
Tabla 8. Requisitos Funcionales 47
Tabla 9. Requisitos No Funcionales de interfaz 49
Tabla 10. Requisitos No Funcionales de seguridad 49
Tabla 11. Requisitos Inversos 50
Tabla 12. Formato de las pruebas del software 52
Tabla 13. PS.01 Representación de las vistas **53**
Tabla 14. PS.02 Proceso de subasta 54
Tabla 15. PS.03 Proceso de carteo 55
Tabla 16. PS.04 Proceso de ayudas 56
Tabla 17. PS.05 Finalización de las manos 57
Tabla 18. PS.06 Opciones cambio de juego 57
Tabla 19. Puntos de honor 61
Tabla 20. Puntuación contrato parcial 63
Tabla 21. Puntuación contrato manga 63
Tabla 22. Puntuación contrato slam 63
Tabla 23. Resultado de las pruebas de software 109
Tabla 24. Costes de personal 122
Tabla 25. Activos amortizables 123
Tabla 26. Costes indirectos 123
Tabla 27. Costes directos 124
Tabla 28. Costes de contratación 124
Tabla 29. Presupuesto inicial 125
Tabla 30. Estimación inicial de cuentas 125
Tabla 31. Presupuesto final 126
Tabla 32. Cuenta final de resultados 126

ÍNDICE DE FIGURAS

Figura 1. Bridge durante los Juegos Olímpicos de Beijing 2008 16
Figura 3. Tutorial de Learn to Play Bridge 22
Figura 2. Ejemplo de partida Learn to Play Bridge 22
Figura 4. BBO 23
Figura 5. BBO application 23
Figura 6. Bridge Master 24
Figura 7. Jack Bridge 25
Figura 8. Blue Chip miniBridge 25
Figura 9. Bridge Baron 26
Figura 10. Q-Plus Bridge 27
Figura 11.Subasta en Fun Bridge 27
Figura 12. Micro MiniBridge 12 28
Figura 13. GIB 29
Figura 14. DealMaster Pro 37
Figura 15. Fichero CSV de *DealMaster Pro* 37
Figura 16. Fichero de texto de las ayudas 38
Figura 17. Diagrama de casos de uso 39
Figura 18. Diagrama MVC 51
Figura 19. Diagrama de componentes del Software 66
Figura 20. Pantalla Bienvenida 68
Figura 21. VistaMenú 68
Figura 22. VistaTutorial 68
Figura 23. VistaPartida 69
Figura 24. Diagrama de clases del componente Vista 69
Figura 25. Diagrama de clase del subcomponente *VistaPartida* 70
Figura 26. Diagrama de clases del subcomponente *ControladorJuego* 1. 72
Figura 27. Diagrama de clases del subcomponente *ControladorJuego* 2. 72
Figura 28. Diagrama de clases del componente Controlador 74
Figura 29. Diseño de las cartas 77
Figura 30. Diseño del *bidding box* 78
Figura 31. Diseño de las voces del *bidding box* 78
Figura 32. Diagrama de secuencia CU.01 79
Figura 33. Diagrama de secuencia CU.02 Reparto 80
Figura 34. Diagrama de secuencia CU.02 Carteo 81
Figura 35. Diagrama de secuencia CU.02 Siguientes repartos y salida 83
Figura 36. Diagrama de secuencia CU.03 Reparto 84
Figura 37. Diagrama de secuencia CU.03 Subasta 85
Figura 38. Diagrama de secuencia CU.03 Carteo 87
Figura 39. Diagrama de secuencia CU.03 Siguientes repartos y salida 88
Figura 40. Disposicion de JFrame y JPanel 90
Figura 41. Disposición de las caras de las cartas 91
Figura 42. Disposición de los dorsos de las cartas 91

Figura 43. Disposición de las voces sobre la mesa 91
Figura 44. Herramienta *Kuler* 92
Figura 45. Disposición del *bidding box* 92
Figura 46. Disposición de las cartas jugadas sobre el panel mesa 95
Figura 47. Panel menú de juego 96
Figura 48. Vista general de la partida miniBridge 96
Figura 49. Vista general de la partida de Bridge 97
Figura 50. Photoshop creando los tutoriales 98
Figura 51. Vista general del tutorial 99
Figura 52. Paneles de casos de uso 100
Figura 53. Vista general de VistaMenú 101
Figura 54. Vista general PantallaBienvenida. 101
Figura 55. Modelo en cascada 117
Figura 56. Diagrama de Gantt final 119
Figura 57. Pantalla inicial 127
Figura 58. Menú de juego 127
Figura 59.Tutoriales 128
Figura 60. Partida de miniBridge 128
Figura 61. Partida de Bridge 129
Figura 62. Bidding Box 132
Figura 63. Subasta en una partida real 137
Figura 64: El declarante, el muerto y la carta de salida 138
Figura 65. Tablilla para guardar las manos de una partida 144
Figura 66. Disposición de las parejas por equipos 145

1. INTRODUCCIÓN

En este capítulo se detalla qué es un proceso de ingeniería, las motivaciones, alcances y objetivos del proyecto así como la estructura general que presenta este documento. También incluye un resumen histórico del Bridge y los tecnicismos más importantes para la facilitar la comprensión del presente documento.

1.1 Proceso de ingeniería

Un proceso de ingeniería tiene como finalidad principal mejorar la calidad de los procesos ingenieriles para llevar a cabo un proyecto. Estos procesos sirven de guía, modelo, herramienta y método a los ingenieros durante el proceso de desarrollo del proyecto. Los principales procesos que se llevan a cabo a lo largo de este tipo de procesos, son los siguientes:

- **Análisis**: proceso en el cual se analizan los objetivos del proyecto, las tecnologías aplicables, los componentes principales, los casos de uso y los requisitos del proyecto.
- **Diseño**: proceso a partir del cual se eligen las herramientas a utilizar y se diseñan los componentes principales en función del proceso de análisis.
- **Implementación**: proceso dónde se desarrolla e implementa todo el proceso de diseño.
- **Pruebas**: proceso en el que se ejecutan las pruebas para la detección de errores en los componentes para poder ser corregidos.
- **Gestión del proyecto**: proceso en el cual se analizan y se detallan los análisis previos y resultados posteriores, tanto del presupuesto económico, como del plan temporal de los procesos del proyecto.
- **Documentación**: proceso en el cual se documenta todo el proyecto, para reflejar todo el proceso de ingeniería, así como utensilio para futuros estudios o mejoras.

Día a día aumenta la demanda de proyectos de *software* como herramientas tanto de trabajo, control o entretenimiento entre otros (1). Con ellos se pretende reducir costes, tanto humanos como materiales intentando mejorar la eficiencia. El proceso de ingeniería se convierte en una herramienta muy útil para realizar proyectos de *software*.

1.2 Motivación.

El presente proyecto tiene como idea principal aplicar el proceso de ingeniería a un entorno concreto, el desarrollo de un *software*. Éste tendrá como finalidad el aprendizaje por ordenador de una materia compleja, como es el juego de mesa conocido como "Bridge". De este modo se permitirá a cualquier usuario iniciarse y familiarizarse con el juego de cartas Bridge y aprender a jugar desde un nivel nulo a un nivel básico.

Este proyecto se fundamenta en la base de que en España hay un desconocimiento importante sobre este juego. Varios son los motivos que hacen que sea poco conocido. El primer motivo a destacar es la procedencia del Bridge, es un juego extranjero y relativamente moderno, sección 1.3. A su vez se destaca que es un juego complicado que requiere una gran implicación durante la etapa de aprendizaje, cabiendo resaltar que, debido a la gran minoría española jugadora, el acceso a este aprendizaje es difícil. El último de los motivos más importantes, es la poca inversión para la difusión del juego y la acogida de nuevos jugadores por parte de organismos españoles. Por ello se presenta interesante crear un sistema que permita mostrar el juego de una manera accesible, sencilla y atractiva.

Este sistema será implementado mediante un *software*, de este modo se facilita enormemente la iniciación a este maravilloso juego, ya que, a día de hoy, existe gran facilidad para el acceso a un PC. Con la creación de un *software* de aprendizaje se elimina la necesidad de terceras personas así como la necesidad de disponer de los materiales físicos necesarios para su desarrollo, que dificultan de un modo u otro el interés y el acceso a este desconocido juego. Además, se recalca que la sociedad tiene poca propensión a iniciarse en actividades de alta implicación si se desconocen las características de dicha actividad. Con estas ventajas el *software* se convierte en la mejor herramienta para dar a conocer el Bridge.

1.3 Introducción histórica al Bridge

Aunque el origen histórico del Bridge pueda fecharse a finales del siglo XVI, e incluso pueda tener varias raíces hispanas, la realidad es que el juego, tal y como se conoce en nuestros días, tiene un claro origen en el *Whist*. El *Whist* era un juego de cartas de entretenimiento popular que se desarrolló considerablemente tras su exportación desde las islas británicas hacia el resto del mundo, y que aún sufrirá una serie de cambios sustanciales hasta llegar al Bridge actual.

Uno de los cambios más importantes fue debido, posiblemente, a la casualidad de que un jugador se viera obligado a faltar a su cita, que dio lugar al conocido *Dummy Whist*, donde aparece la figura del muerto *(dummy)*.

A finales del siglo XIX comienza a emplearse el nombre del Bridge, que según Miguel Mestanza (ex presidente de la Asociación Española de Bridge), es el eslabón que entronca toda una serie de juegos como el Bridge, la Brisca en España, el *Biritch* en Rusia o la *Bríscola* en Italia, cuyos orígenes están cerca de nuestro popular Tute.

Durante todo el siglo XIX empiezan a proliferar los clubes de *Whist*, tanto en Gran Bretaña como en otros países, con lo que el juego recibe un gran empujón hacia la popularidad. A su vez, en esta época, tienen lugar los primeros estudios serios y los primeros nombres de jugadas, así como la fundación del Club más importante del mundo, el *Portland Whist Club* de Londres.

Lo que hoy en día se conoce como Bridge, debe su paternidad a un multimillonario norteamericano llamado Harold S. Vanderbilt, quien durante un crucero frente a las costas panameñas, decide crear el primer reglamento de Bridge. Las reglas creadas por Vanderbilt fueron aceptadas en todos los países donde se practicaba el juego, con lo que pasó a tener una categoría y una fuerza de los que antes carecía.

Vanderbilt fue a su vez el impulsor de lo que sería la Federación Mundial de Bridge que más tarde celebraría el primer Campeonato del Mundo, y que en 1960 organizaría las primeras olimpiadas de bridge coincidiendo con las olimpiadas deportivas (2).

En marzo de 1999, en una reunión celebrada en Seúl, el Comité Olímpico Internacional (COI) reconoció a la Federación Mundial de Bridge (WBF), con Juan Antonio Samaranch en la presidencia, como Federación Olímpica y otorgando al Bridge el mismo status deportivo que otros deportes como el golf, el squash, el ajedrez, karate, etc. Es decir, modalidades que se consideran deporte a todos los efectos aunque no participan en los Juegos Olímpicos.

Cada cuatro años, entre julio y agosto, tienen lugar los Juegos Olímpicos en una sede olímpica diferente. A su vez, en la misma sede en agosto tienen lugar los Juegos Paraolímpicos y en último lugar, en septiembre, los Juegos Olímpicos de la Mente, como se puede apreciar en la Figura 1. En los Juegos Olímpicos de la Mente tienen cita tres disciplinas, tres juegos, el Ajedrez, el Go y el Bridge.

Figura 1. Bridge durante los Juegos Olímpicos de Beijing 2008

A día de hoy, el Bridge se practica en todo el ámbito mundial, más de 80 millones de aficionados disfrutan de él, más de 125 países cuentan con una Federación Nacional de Bridge, o entidad análoga, que lo regula. Además se destaca que la literatura de Bridge compite con la de ajedrez y se ha generalizado su introducción en escuelas y universidades fuera de España. El número de jugadores federados en España ronda los 5.000 y el de personas que lo practican se estima en 35.000 (3).

Los objetivos y las reglas de juego se pueden consultar en el Anexo III de este documento.

1.4 Tecnicismos utilizados

A continuación se explican algunos tecnicismos del Bridge para mejorar la comprensión de este documento.

- **Asistir al palo**: Es una de las reglas principales del Bridge, en la cual un jugador está obligado a jugar una carta del **palo** con el que se empezó la **baza**, siempre y cuando se tenga una o más cartas de dicho palo en su haber.

- **Baza**: Un conjunto de cuatro cartas jugadas, una de cada jugador, durante el **carteo.**

- ***Bidding box*** (caja de voces): Como su propio nombre indica, es el utensilio donde se colocan todas las **voces** para llevar a cabo la **subasta.**

- **Carteo**: Es la segunda fase de una partida. Se juegan las cartas.

- **Contrato**: Número de **bazas** y **palo de triunfo** contratados en la **subasta**.

- **Contrato Parcial**: Contrato comprendido entre los niveles 1 y 3, sin incluir la voz de 3**ST.**

- **Convenciones**: Acuerdo previo a la partida entre compañeros de una misma pareja para jugar un mismo **sistema de juego**.

- **Cruzar honores**: Es una buena jugada que permite que los **honores** de los adversarios sean anulados durante una **baza**.

- ***Dealer* o dador**: Es el jugador que comienza la **subasta**.

- **Declarante**: Aquel jugador que ha "ganado" la **subasta** y tiene que cumplir el **contrato**. La pareja contraria es la pareja defensora.

- **Doblo**: Una de las voces especiales del ***bidding box***.

- **Fallo y descarte**: es una mala jugada en la cual se cede, en la mayoría de los casos, una **baza** a los contrarios.

- **Honores**: Son las cartas más altas de cada **palo**, As, K, Q y J.

- ***Impasse***: **Honores** de las cartas de un jugador de un mismo **palo** con una diferencia de rango de dos o más puntos.

- **Jugador Norte, Sur, Este u Oeste**: Disposición técnica de la orientación de los cuatro jugadores en la mesa durante la partida.

- **Manga**: Nivel de subasta en el que aumentan las puntuaciones, comprendido entre las **voces** de **3 ST** y **5 ♠**, ambas incluidas, exceptuando la voces de 4♦ y 4♣.

- **Mano**: Es un reparto de cartas.

- **MiniBridge**: Es un tipo de partida en el que únicamente se juegan las cartas, es decir, solo hay **carteo**. Se utiliza como método didáctico.

- **Muerto**: aquel jugador compañero del **declarante** que no juega sus cartas durante el **carteo,** desplegándolas sobre la mesa. Estas cartas expuestas sobre la mesa, a la vista de todos, las juega su compañero, el declarante.

- **Multa**: Una vez terminado el **carteo**, si el recuento de **bazas** de la pareja **declarante** no ha alcanzado el número acordado en el **contrato**, serán tantas multas como sea la diferencia. Las multas son puntos para la pareja defensora.

- **Palo**: Cada uno de los cuatro símbolos de la baraja, picas (♠), corazones (♥), diamantes (♦) y tréboles (♣).

- **Palos mayores**: Picas y corazones.

- **Palos menores**: Diamantes y tréboles.

- **Puesta en mano**: Jugada durante el **carteo** en el que se cede la **baza** pudiendo, no hacerlo, para obtener algún tipo de beneficio.

- **Redoblo:** Es una de las **voces** especiales del ***bidding box***.

- **Renuncio**: Infracción de un jugador, cometida por no **asistir al palo** pudiendo haberlo hecho.

- **Secuencia**: **Honores** de un mismo palo en la mano de un jugador (mínimo tres).

- **Semisecuencia**: Dos **honores** consecutivos y el tercer honor es el siguiente no consecutivo.

- **Sin Triunfo o ST:** Es un **palo** artificial en el Bridge en el cual no **triunfa** ningún palo.

- **Sistema Natural:** Sistema de comunicación llevado a cabo por una pareja durante la **subasta** en la que las **voces** dadas se refieren únicamente al **palo** dado.

- **Slam:** Nivel de **subasta** comprendido entre los niveles 6 y 7, ambos incluidos.

- **Subasta**: Es la primera fase de una partida. No se juegan las cartas, se juegan las voces.

- **Triunfo o Palo de Triunfo**: **Palo** que domina sobre los demás durante el carteo de una mano.

- **Voz**: Intervención de un jugador durante la **subasta**.

- **Vulnerabilidad**: cuando una pareja es vulnerable la puntuación asignada es diferente para cierto tipo de **contratos** y resultados.

1.5 Objetivo del proyecto

El objetivo de este proyecto es analizar, diseñar e implementar un *software* de aprendizaje, en castellano, que permita al usuario iniciarse y familiarizarse con el Bridge mediante tutoriales y partidas interactivas.

1.6 Alcance del Proyecto

Este proyecto está destinado a usuarios de todas las edades y generaciones que quieran iniciarse al Bridge y familiarizarse con las partidas. Por lo que se diseñará un sistema de juego con un nivel de dificultad entre básico e intermedio. Con este

software de juego no se pretende dificultar la partida sino facilitar y guiar al usuario a llevar a cabo un buen juego.

El *software* se implementará en castellano, con el tipo de partida miniBridge, por motivos didácticos, y partida de Bridge. A su vez facilitará tutoriales para el conocimiento y el aprendizaje del juego. Todos estos requisitos prestarán atención a crear una interfaz que resulte sencilla y familiar a un aprendiz de este juego.

1.7 Organización del presente documento

Este documento presenta la siguiente estructura:

- En este capítulo se ha introducido el problema a tratar, los objetivos del proyecto y el alcance del mismo sobre la solución propuesta.

- En el capítulo 2, Análisis, se analizan los programas de Bridge más importantes, las tecnologías más relevantes de cara a desarrollar los objetivos del proyecto, así como los casos de uso, los requisitos funcionales, no funcionales, e inversos del mismo. Por último se especifica el plan de pruebas del proyecto y un estudio técnico de Bridge.

- En el capítulo 3, Diseño, en función del análisis y de los requisitos, reflejados en el capítulo 2, se llevará a cabo el diseño del mismo. Se detallarán los diagramas de componentes, de clases y de secuencia pertinentes.

- En el capítulo 4, Implementación, se recogen y se detallan los aspectos más importantes del proceso de implementación del *software*.

- En el capítulo 5, Conclusiones y líneas futuras, se aportan comentarios personales del autor sobre el proyecto de todo el proceso de elaboración del mismo y las líneas futuras en las que se puede mejorar para incrementar su funcionalidad.

- Posteriormente se incluyen las referencias utilizadas para el desarrollo del proyecto y su documentación. Al final del documento se incluyen los diferentes anexos, Anexo I, Gestión del Proyecto, Anexo II, Manual de usuario y por último el Anexo III, Introducción a la partida de Bridge.

2. ANÁLISIS

Una vez definidos en la introducción los objetivos del proyecto, se analizarán en este capítulo los principales aspectos del mismo. Por ello se procederá a detallar, entre otros, un estudio de los principales programas de Bridge (sección 2.1 y 2.2), un análisis de las tecnologías aplicables (sección 2.3) y un estudio de los ficheros necesarios para el *software* (secciones 2.4 y 2.5). Posteriormente se detallan los casos de uso que llevará a cabo el programa (sección 2.6) y los requisitos del mismo (sección 2.7).

Finalmente se procederá a realizar un análisis de la arquitectura preliminar de los componentes del proyecto así como un análisis detallado del plan de pruebas (secciones 2.8 y 2.9 respectivamente). Además, se ha representado parcialmente, un análisis de la técnica de juego de nivel intermedio (sección 2.10).

Se aclara que, para el análisis de los programas de Bridge que no son gratuitos, se solicitó a la empresa una licencia gratuita de dos días de duración, o se descargó la variante *demo*.

2.1 Estado del arte

En la actualidad existen numerosos programas dedicados al Bridge, la gran mayoría provenientes de EE.UU, país con mayor número de jugadores. Todos ellos con unas características diferentes, pero teniendo como idea principal jugar partidas para mejorar el nivel de juego o como mero entretenimiento.

En este apartado se procederá a hacer un estudio de los programas de Bridge que ya existen en el mercado. Primero se presentará cada uno de ellos y posteriormente se refleja un estudio en una tabla con las diferentes características de dichos programas.

2.1.1 Learn To Play Bridge

Learn To Play Bridge (4), es un *software* de Bridge únicamente disponible en inglés que fue diseñado por la casa *Microsoft* para la *American Contract Bridge League* (A.C.B.L) con la posibilidad de ejecutarse únicamente en soportes *Microsoft Windows*. Pero el año pasado implementaron una versión para ordenadores *Mac*.

Es un *software* con fines didácticos para jugadores con un nivel de juego intermedio para perfeccionar y afianzar las bases ya aprendidas así como aprender

nuevas jugadas y convenciones. La parte didáctica fue creada con la ayuda de muchos jugadores americanos de alto nivel por lo que la teoría aplicada es de gran calidad.

Este programa carece de la posibilidad de jugar partidas, factor muy negativo para el aprendizaje. En su lugar, como medios didácticos, ofrece un *e-book*, Figura 2, y la representación de diferentes ejemplos con partidas inactivas, Figura 3.

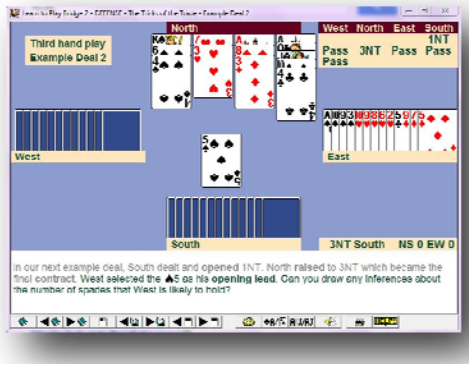


Figura 3. Ejemplo de partida Learn to Play Bridge

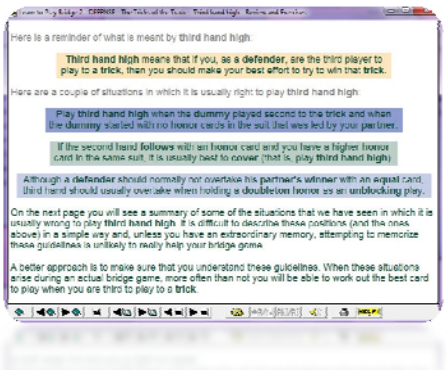


Figura 2. Tutorial de Learn to Play Bridge

2.1.2 Bridge Base Online (B.B.O)

BBO (5) es un *software* de Bridge, disponible en todos los idiomas, diseñado por la casa *Microsoft*. Este programa puede ejecutarse en todos los dispositivos a excepción de dispositivos tipo "tableta" y dispositivos avanzados tipo "*smartphone*".

BBO fue diseñado para permitir jugar de partidas y torneos únicamente *online*, las 24 horas del día y como concepto de entretenimiento para el usuario, por lo que no es un programa didáctico. Proporciona partidas libres y torneos gratuitos o de pago. Recientemente este programa incorporó la posibilidad de ser espectador de partidas y esto creó la idea del *Vugraph*. *Vugraph* retransmite las partidas a tiempo real de los torneos más importantes de todo el mundo para que los usuarios puedan disfrutar de ellas. Son comentadas en chat por profesionales y espectadores.

Este programa es el más usado para jugar partidas Online en todo el mundo, con más de 100.000 miembros, con una media de 7.000 jugadores conectados al mismo tiempo y una media de entrada de usuarios de 20.000 al día (5).

La interfaz que ofrece este juego es muy esquemática y concentrada, como se observa en la Figura 4, por lo que se requieren conocimientos intermedios de juego para poder disfrutar este programa correctamente.

Figura 4. BBO

2.1.3 BBO IOS & Android application

BBO application (5) es un *software* de Bridge, disponible exclusivamente en inglés, diseñado por *Microsoft*. Esta aplicación es exactamente igual que el *BBO*, (sección 2.1.2), pero con dos excepciones. La primera excepción es que únicamente es ejecutable en dispositivos tipo "tableta" y dispositivos con capacidad avanzada "*smartphones*", *Mac* y *Android*. La segunda excepción es que no incorpora *Vugraph*. Con esta aplicación se participa en las mesas de juego del programa *BBO* normal.

Figura 5. BBO application

2.1.4 Bridge Master 2000

Bridge Master 2000 (6), es un *software* de Bridge diseñado por la casa *Microsoft*. Está disponible en todos los idiomas, pero uno a elegir desde el principio.

Este programa está diseñado para enseñar y perfeccionar únicamente estrategias de carteo, ofreciendo únicamente partidas sin conexión, con manos ya

repartidas y la subasta representada esquemáticamente en la pantalla. Tiene el mismo motor gráfico que *BBO*, sección 2.1.2.

Este es el programa más completo y perfecto de todos los que hay para mejorar el carteo, pero para ello se requieren conocimientos previos intermedios para poder desarrollar y entender la partida correctamente.

El tipo de partidas que ofrecen son ejercicios prediseñados en forma de manos con un objetivo en bazas (carteo), facilitando ayudas una vez finalizado cada ejercicio si el usuario así lo solicita. Se puede observar un ejemplo en la Figura 6.

Como característica principal se recalca que posee un análisis perfecto de carteo así como de análisis de errores cometidos por el usuario durante el mismo, interpretándolos e intercambiado las cartas de los oponentes sin que el usuario lo sepa y de este modo no pueda cumplir los objetivos. Con ello consigue que si el carteo ha sido incorrecto, el factor suerte ya no pueda intervenir. Se destaca que este programa no indica el o los momentos exactos donde se ha equivocado el usuario, sino que explica por texto cómo se debe jugar la mano correctamente y por qué motivos.

Figura 6. Bridge Master

2.1.5 Jack Bridge

Jack Bridge (7) es un programa diseñado por y para jugadores de Bridge con un objetivo didáctico pero a su vez ofrece partidas *online*. Está disponible en los idiomas principales exceptuando el español.

El gran punto fuerte de este programa es su sistema de aprendizaje y enseñanza de muy buena calidad. Ofrece ayudas de juego durante la subasta y ejemplos muy buenos para mejorar el nivel.

Para poder utilizar y disfrutar correctamente este programa se necesitan unos niveles de conocimientos previos intermedios. Además, la interfaz gráfica de la que dispone el programa es muy antigua, Figura 7. Este *software* ofrece la posibilidad de

jugar partidas y torneos *online*, pero no hay gran número de jugadores debido la gran mayoría juega online en *BBO* (sección 2.1.2).

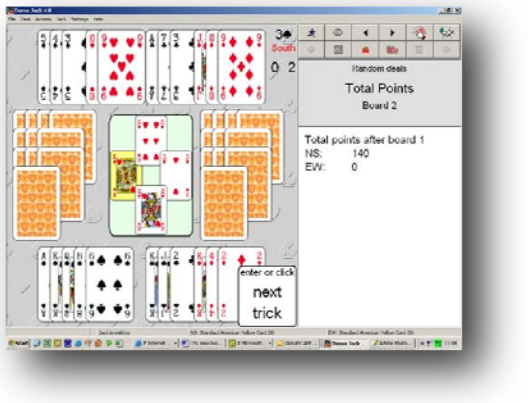


Figura 7. Jack Bridge

2.1.6 Blue Chip MiniBridge

Blue Chip MiniBridge (8) es un *software* diseñado en EE.UU exclusivamente para sistemas operativos *Windows* y disponible únicamente en inglés. Es un programa muy antiguo, lento y sencillo el cual permite jugar partidas de miniBridge con manos repartidas aleatoriamente, como se observa en la Figura 8.

Carece de tutoriales y ayudas, por lo que es un programa para familiarizar al usuario con el carteo sin necesidad de conocimientos previos para poder jugar. Pero utilizando este programa no se aprende correctamente a jugar partidas de Bridge completas ni a entender sus objetivos.

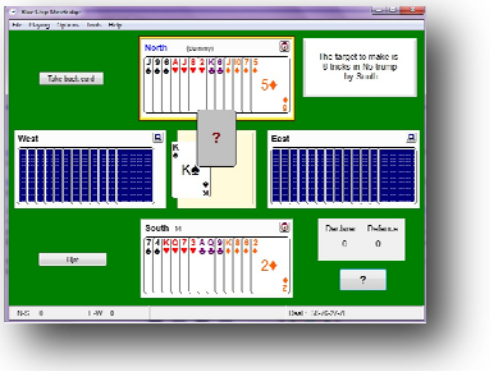


Figura 8. Blue Chip miniBridge

2.1.7 Bridge Baron 22

Bridge Baron (9) es un *software* de bridge estadounidense disponible en los principales idiomas incluyendo el español. Puede ejecutarse en cualquier tipo de sistema operativo, *Windows*, *Mac*, *IOS* y *Android*. Este programa tiene como objetivo mejorar las técnicas Bridge del usuario con la posibilidad de jugar partidas y torneos contra la máquina como mero entretenimiento, Figura 9.

Con este programa se puede jugar al miniBridge, Bridge y torneos. Tanto con manos preparadas como aleatorias.

Una de las principales características es la posibilidad de jugar la modalidad torneos contra la máquina sin la necesidad de conexión *online*. Pero no son torneos cualquiera, sino torneos profesionales ya realizados en EE.UU en los cuales el usuario compite contra los resultados que han obtenido los otros jugadores que jugaron físicamente el torneo.

Figura 9. Bridge Baron

2.1.8 Q-Plus Bridge

Q-Plus Bridge (10) es un *software* de bridge alemán disponible en los idiomas principales exceptuando el español. También existe el programa en versión para dispositivos de capacidad avanzada tipo "*smartphones*" con sistema operativo *Windows* cuyo nombre es *T-Plus Bridge*.

Es un programa destinado a mejorar el nivel de juego del usuario. También es utilizado como entretenimiento. Ofrece únicamente partidas de Bridge y sin conexión *online*. Tiene una interfaz moderna exceptuando la fase de subasta que es antigua, Figura 10. El sistema para "*smartphones* "es muy antiguo, reducido y sistemático. Para

jugar y disfrutar el programa se necesitan conocimientos intermedios de juego y las ayudas no tienen lugar durante la partida sino que son posteriores y complicadas de entender. Ofrece un tutorial en la página web disponible en los mismos idiomas que posee el juego.

Figura 10. Q-Plus Bridge

2.1.9 Fun Bridge

Fun Bridge (11) es un *software* de Bridge estadounidense disponible en los idiomas principales, incluyendo el español. Este programa se puede ejecutar tanto en *Windows* y dispositivos *Mac* como ordenadores *Mac*, *iPad* e *iPhone*.

Fun Bridge tiene como propósito principal entretener al usuario. Es un juego que se descarga gratuitamente pero se juega exclusivamente *online* pagando una cuota mensual. El usuario crea un cuenta *online* para jugar partidas y torneos y según vayan siendo sus resultados subirá o bajara en el ranking general del programa.

Este programa ofrece ejercicios que no intervienen en la puntuación para el ranking. Estos ejercicios ofrecen consejos y ayudas, pero únicamente durante la subasta. Para poder disfrutarlo y entender los consejos y ayudas, el usuario ha de tener unas bases intermedias de juego.

La principal característica de este *software* es su interfaz. Posee una interfaz moderna y atractiva, Figuras 11 y 12. Por este motivo *Fun Bridge*, va consiguiendo poco a poco más adeptos.

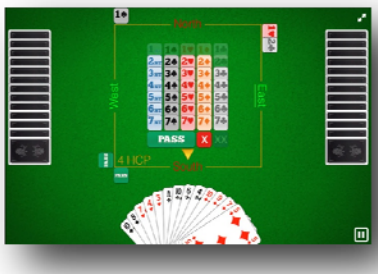


Figura 11.Subasta en Fun Bridge

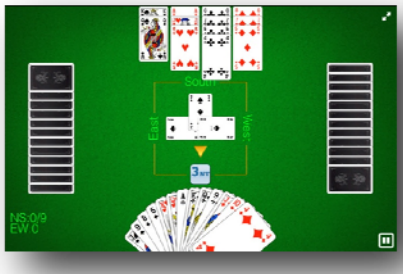


Figura 12. Carteo en Fun Bridge

2.1.10 Micro MiniBridge 12

MicroMiniBridge 12 (12) es un *software* de Bridge estadounidense disponible únicamente en inglés. Fue diseñado con objetivos didácticos y como entretenimiento. Para poder utilizar este programa no se necesita ningún tipo de conocimiento. Pero carece de tutoriales, por lo que el usuario ha de ir analizando e interpretando las características del juego por sí mismo. Cosa que resulta bastante lenta y complicada. Solo aporta consejos, no ayudas, durante la subasta y nada durante el carteo.

Es un programa que ofrece partidas únicamente sin conexión de miniBridge y Bridge con ejemplos muy didácticos y de gran nivel, como se observa en la Figura 13. Como defectos a destacar son la interfaz, que es muy antigua y esquemática y la lentitud del programa a la hora de jugar partidas. Estos motivos no reflejan un juego dinámico y atractivo.

Figura 12. Micro MiniBridge 12

2.1.11 GIB

GIB *(Ginsberg's Intelligent Bridgeplayer)* (13) es un *software* de Bridge estadounidense disponible en los principales idiomas incluyendo el español, pudiendo ejecutarse en sistemas operativos *Windows* y *Linux*.

Se caracteriza por tener una potencia de cálculo y juegos muy buena, pero carece de ayudas y propone partidas muy complicadas. Se requiere un buen nivel de juego para enfrentarse a este *software*. La interfaz de juego es poco atractiva como se puede observar en la Figura 13.

Figura 13. GIB

2.2 Comparativa de los programas presentados

A continuación se presenta una tabla resumen, (Tablas 1 y 2), de los programas de Bridge anteriormente citados y analizados, calificando sus características en función de diversos parámetros que se detallan previamente. Recuérdense los tecnicismos especificados en la sección 1.4 para la comprensión correcta de este apartado.

- **Estética:**

  En esta casilla de la tabla se valoran los diferentes aspectos técnicos para diferenciar entre una interfaz antigua, intermedia y moderna. Se hará una media de todos los factores que se detallan a continuación.

  El principal factor técnico a tener en cuenta es el estilo de baraja. Otros factores importantes son la disposición de las cartas tanto del muerto como del usuario en la interfaz y la representación de la subasta. En primer lugar, diferenciando el estilo de la baraja, existen cuatro tipos de simbologías diferentes, inglesa, francesa, con el logotipo del palo y esquemática. La simbología inglesa puede tener dos o cuatro números, o letras, por carta. Las cartas con cuatro números están diseñadas para zurdos, luego, no hace falta esa representación en el *software*. Si la carta dispone de dos números, la estética de la carta será moderna. Si por el contrario son cuatro números, la estética será intermedia. La simbología francesa dispone de figuras más antiguas que la baraja inglesa, por ello, tanto si posee dos o cuatro números o letras por carta, las cartas tendrán una estética antigua. Si las cartas tienen únicamente representado el logotipo del palo al que pertenecen además de los números o letras, la estética será intermedia. Y por último, si la representación de las cartas es esquemática y no real, la estética será antigua.

El segundo factor, referido a la disposición de las cartas sobre la interfaz, también es muy importante visualmente. En primer lugar estudiaremos las diferentes posiciones en las que se pueden representar las cartas del usuario, es decir, las cartas del jugador Sur. La primera de las posiciones, y más común, es no separar las cartas entre unos palos y otros. Si la mano de Sur no posee una separación entre palos, el valor estético será indiferente, si por el contrario existe tal separación, la estética será antigua. La segunda de las posiciones, es cuánto se tapan unas cartas a otras. Si la separación entre cartas es menor del 20% del ancho, la estética será intermedia. Entre 20% y 50% la estética será moderna y valores superiores a 50% intermedia. Para la disposición de las cartas del muerto se valora si la mano está separada por columnas de palos o no. Si las cartas están divididas en columnas la estética será moderna, por el contrario será antigua con todas las posiciones atribuibles a las cartas de Sur, anteriormente especificadas, exceptuando que será indiferente cuánto se tapan unas cartas a otras.

Por último factor a estudiar de la estética general del juego es la representación de la subasta. Para ello diferenciaremos entre *bidding box* y *voces* ya subastadas. Si el *bidding box* tiene una representación esquemática, sin representar todas las voces posibles directamente, la estética será antigua, si por el contrario se representa completo, el valor estético añadido será indiferente. Si las voces ya dadas se representan esquemáticamente en una tabla, la estética de la subasta será antigua, si por el contrario se dan voces representándose éstas sobre el tapete, la estética será moderna. Cabe recalcar que ningún juego pone sobre el tapete una representación real de las voces.

Además de estos factores, se evalúan otros aspectos no mesurables como son el aspecto general del juego, como los tapetes, el fondo, combinación de colores, menús, etc.

- **Nivel de juego máquina**

  En esta casilla de la tabla se califica el estudio del nivel de juego de la máquina en función de su comportamiento durante el juego. Para ello estudiaremos las diferentes jugadas que realiza el programa. Se hará una media de todos los factores que se detallan a continuación.

  Si el programa analiza los errores cometidos tanto en la subasta como en el carteo, éste programa tendrá un nivel muy bueno, si no es así el nivel de es intermedio. Si el programa, una vez identificados los errores toma medidas para que no se puedan cumplir las bazas, el nivel del juego máquina es perfecto. Si por el contrario el programa no toma ninguna medida, el nivel de juego no se verá modificado.

Si la salida inicial para proceder al carteo es correcta con el sistema natural, y con idea de transmitir información a los jugadores, el programa estará dotado de un nivel de juego máquina muy bueno. Si este programa tiene algún fallo implementando este sistema, tendrá un nivel bueno, si por el contrario no toma en consecuencia la salida inicial, el nivel de juego del programa será malo.

Todos los programas deben evitar la jugada "fallo y descarte", jugada muy común. Si evita la jugada, el nivel de juego no se ve alterado, pero si no evalúa este tipo de jugadas, el nivel del programa es malo.

Por último se evalúan las jugadas más complicadas, la de "puesta en mano" y "cruzar honores". Si cumplen estas jugadas el programa tendrá un nivel bueno, por el contrario el nivel de juego será intermedio.

Nombre	Nivel de conocimientos	Idiomas	Partidas sin conexión Online	Estética	Manos preparadas	Partida Online	Ayudas	Nivel de juego máquina	Precio
Nombre del Programa sometido a estudio	Nivel de conocimientos previos para jugar	Idiomas disponibles en el programa	Tipos de partidas disponibles sin conexión	-Moderna - Intermedia -Antigua	Posibilidad de jugar manos preparadas	Jugar partidas Online contra otros jugadores	Tipos de Ayudas que facilita el programa	Nivel de juego del programa en las partidas sin conexión	Cantidad económica aproximada a la que está disponible el juego

Tabla 1. Formato de tabla de propiedades de los programas de bridge estudiados

Nombre	Nivel de conocimientos	Idiomas	Partidas sin conexión	Estética	Manos preparadas	Partida Online	Ayudas	Nivel de juego máquina	Precio
Learn to Play Bridge (Microsoft)	Intermedio	Inglés	Ninguna	Antigua	Sí	No	**E-book** Ejemplos	Nulo	Gratuito
BBO	Intermedio	Todos	Ninguna	Intermedia	No	Sí	No	Nulo	Opción de pago.
BBO IOS application	Intermedio	Inglés	Ninguna	Intermedia	No	Sí	No	Nulo	Opción de Pago
Bridge Master (Microsoft)	[Intermedio-Muy Avanzado]	Uno a elegir	Carteo	Intermedia	Sí	No	Posteriores a la mano	Perfecto	[90-300]€
Jack	Intermedio	Principales Sin Español	Bridge	Antigua	Sí	Sí	Durante Subasta	Muy bueno	80€
Blue Chip miniBridge	Nulo	Inglés	Ambas	Antigua	No	No	No	Malo	Gratuito
Bridge Baron	Intermedio	Principales	Bridge MiniBridge	Intermedia	Opcional	Sí	No	Intermedio	120€
Q-Plus Bridge	Intermedio	Principales Sin Español	Bridge	Moderna	Sí	No	Despues del carteo	Bueno	Gratuito
Fun Bridge	Intermedio	Pocos	Ninguna	Moderna	No	Sí	Durante subasta	Bueno	Cuota Mensual
MicroMiniBridge 12	Nulos Básicos	Inglés	MiniBridge Bridge	Antigua	Sí	No	Durante subasta	Básico	Gratuito
GiB	Intermedio-Avanzado	Principales	Bridge	Antigua	No	No	No	Bueno	60€

Tabla 2. Tabla de propiedades de los programas de Bridge estudiados

A raíz de este análisis se obtiene la conclusión de que no existe ningún programa que enseñe a jugar al Bridge con una enseñanza interactiva. Además, la ayuda que aportan estos programas a la iniciación es nula o limitada a ciertas partes del juego. Cabiendo destacar que ninguno de ellos introduce a jugar al Bridge desde un conocimiento nulo del juego atrayendo así más jugadores.

Los programas que ofrecen ayudas durante el juego, únicamente reflejan las mismas durante la fase de subasta de la mano, ninguno de ellos ayuda u orienta cómo proceder a un buen carteo durante la partida. El único programa que ofrece ayudas y explicaciones sobre el carteo es el *Bridge Master*, pero las ofrece una vez terminada la mano, si el usuario las solicita. Este es el mejor programa para mejorar el nivel de carteo. Aun siendo este programa el más completo para cartear, sólo da la oportunidad de jugar la modalidad de carteo, pero no es miniBridge porque viene con la subasta representada en una tabla, por lo que el usuario debe tener conocimientos previos sobre el funcionamiento y la teoría de la subasta para poder interpretar correctamente las explicaciones que aporta el programa.

Los otros programas que implementan ayudas durante la partida son el Jack, el *Fun Bridge* y el *GiB*. Estas ayudas requieren niveles previos de conocimientos intermedios para ser interpretadas adecuadamente y están destinadas únicamente a la subasta. Estos programas están orientados a mejorar el nivel de juego del usuario o, como se ha nombrado anteriormente, como herramienta de entretenimiento.

Los únicos programas destinados a la iniciación en el Bridge son el Blue Chip *Mini Bridge* y el *Micro MiniBridge*. Ambos carecen de tutoriales de aprendizaje para conocer el Bridge. El Blue Chip no posee mano preparas, vitales en este juego para la enseñanza. Se recalca también, que ninguno de ellos está disponible en castellano y no tienen como objetivo dar a conocer el Bridge ni representar el juego correctamente.

Se observa a su vez que muchos de los programas están destinados a partidas *online* como plataforma de entretenimiento. Otro factor clave a destacar es que ninguno de ellos representa una subasta fielmente a la realidad. Esto imposibilita que alguien, que nunca haya jugado, entienda la fase de subasta correctamente.

Prestando atención a la estética de la interfaz gráfica de estos programas, cabe resaltar que únicamente dos de ellos implementan una estética moderna. Esto es debido a que la mayoría de los programas de Bridge están destinados a la potencia de cálculo y de juego y no a la representación atractiva de los elementos sobre la interfaz. Además se recalca que son programas antiguos cuyas versiones a lo largo de los años, se han centrado mayoritariamente la incorporación de nuevos contenidos en y la potencia del cálculo anteriormente citada.

Con este estudio se demuestran las motivaciones y objetivos del proyecto (secciones 1.2 y 1.5).

2.3 Análisis de tecnologías

En esta sección se hará un estudio de las tecnologías aplicables para llevar a cabo el proyecto. En el capítulo de diseño se referencia qué tecnologías se utilizaran e implementaran y los motivos por los que han sido escogidas.

Se empezará detallando los tipos de lenguajes de programación y posteriormente se estudiaran los entornos para implementar y compilar estos lenguajes.

2.3.1 Lenguaje de programación C++

C++ (15) es un lenguaje de programación que se planteó como una mejora de las capacidades y funcionalidades del lenguaje ***C***. Ahora es un lenguaje independiente.

Es un lenguaje orientado a objetos, con multitarea entre clases y que permite modularidad. Es un lenguaje muy eficaz usado por muchos programadores en el mundo.

La principal ventaja de *C++* es la potencia de cálculo. Esta característica hace que sea un lenguaje muy solicitado para bases de datos y desarrollos web.

2.3.2 Lenguaje de programación Java

Java (14) es un lenguaje de programación de la casa *Sun Microsystems* desarrollada para la creación de *software* en pequeños dispositivos.

Java es un lenguaje abierto (gratuito) orientado a la creación de objetos con una sintaxis similar a la de ***C++***. Pero respecto a este último, tiene algunas simplificaciones en sus características como son la sobrecarga de operadores, la herencia múltiple, el paso por referencia de parámetros, la gestión de punteros, la liberación de memoria automática y las instrucciones de pre compilación, haciendo este lenguaje más versátil de cara a la programación de objetos.

Otras dos de las principales ventajas que nos ofrece *Java* son el *API* (26) y la otra es que Java es independiente de la plataforma (hardware) y del sistema operativo base (*software*). *API* es una librería o biblioteca de clases ya creadas en Java de gran utilidad para la creación de nuevas clases y métodos.

2.3.3 Lenguaje de programación JavaScript

JavaScript (16) es un lenguaje de programación interpretado, lo que significa que no necesita ser compilado. Proviene de *Java*, por lo tanto también es un lenguaje de programación orientado a objetos. Se utiliza principalmente para la creación de páginas web. No obstante, es un lenguaje muy diferente de *Java*. *JavaScript* aunque es un lenguaje de programación orientado a objetos no tiene herencia, al contrario que *Java*. Estas características hacen que a su vez sea un lenguaje orientado a eventos.

Otra diferencia entre ambos lenguajes es que, mientras con Java podemos crear aplicaciones autónomas como son los *applets* (programas que podemos incluir en las páginas web), *JavaScript* es un lenguaje que se incorpora dentro de la página web, formando parte del código *HTML* sin el que no puede existir.

2.3.4 Lenguaje de programación Groovy

Groovy (17) es un lenguaje de programación orientado a objetos para la plataforma *Java*, como alternativa a este lenguaje porque es más rápido, dinámico y ágil. Además puede usarse como lenguaje de scripting dentro de la plataforma Java.

Es un lenguaje potente para la implementación de aplicaciones en páginas web, sin apenas desarrollar código.

2.4 Dealmaster Pro

Dealmaster Pro (18) es un programa desarrollado en 1996 por la casa *Ward & Sons, Inc*, fundada en 1976 por Edward C. Marzo. Este programa hasta la fecha ha tenido numerosas actualizaciones y mejoras pero únicamente es compatible con sistemas operativos *Windows*.

Fue diseñado como herramienta para profesionales y profesores de Bridge permitiendo repartir manos para torneos o para ejemplos didácticos de una manera fácil y rápida. Estos ejemplos didácticos pueden estar destinados a libros y revistas o a la representación de partidas como es nuestro caso. Se puede observar una representación del programa en la Figura 14.

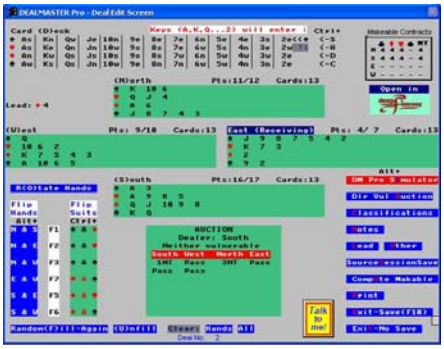


Figura 14. DealMaster Pro

Dealmaster Pro ofrece la posibilidad de exportar el documento creado en varios tipos de formato. Este proyecto se centra en el análisis del formato *CSV* creado por este programa. Este formato representa con letras y números las cartas de este modo:

A K Q J T 9 8 7 6 5 4 3 2

El programa dispone las 52 cartas en una única fila. Las distribución de las manos a sus jugadores lleva el esquema "NESW", es decir, las primeras trece cartas corresponden al jugador Norte, las trece siguientes a Este, las siguientes trece a Sur y las trece últimas corresponden a Oeste. Las manos ya vienen ordenadas en rango y palo, pero separadas por comas entre los diferentes palos. Dentro de las trece cartas de cada jugador, el orden de palos es el siguiente: Picas, Corazones, Diamantes y Tréboles.

A continuación de la carta número 52 y sus correspondientes comillas, se indican el número de mano, el jugador *dealer* y la vulnerabilidad de las parejas. Este último factor no se tendrá en cuenta para el desarrollo de este proyecto. Con este formato, Figura 15, se representa y se sitúa cada carta con un único carácter.

	A	B	C	D	E	F	G	H	I
1	AJT86,"82","AK4","AK8","53","AT6","QJ92","QJT9","KQ942","KQ7","653","72","7","J9543","T87","6543","1","N/-"								

Figura 15. Fichero CSV de *DealMaster Pro*

2.5 Fichero TXT

El sistema de aprendizaje y de ayudas, que proporcionará el *software* durante las partidas, se obtendrán, parte de ellas, de un fichero de texto *(.txt)*. El formato de dicho documento se refleja en la Figura 16.

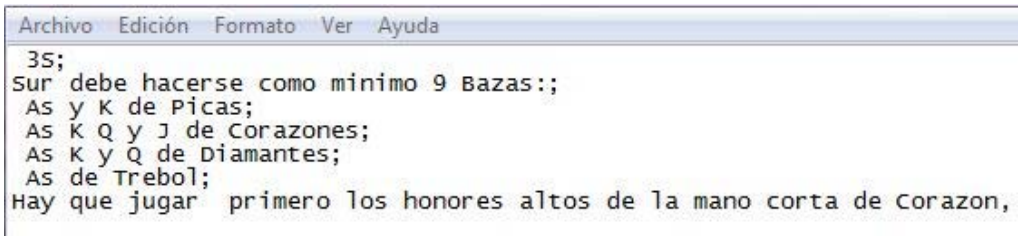


Figura 16. Fichero de texto de las ayudas

La primera fila transmite la información al *software* de cuál es el contrato de juego más adecuado. Las siguientes filas, delimitadas por un *punto y coma (;)*, muestran las ayudas creadas previamente por un profesional del Bridge, para un proceder a un correcto juego. Estas ayudas se representarán por pantalla en el *software*, por ello estos documentos no llevaran ninguna tilde, así no habrá problemas de compilación ni de caracteres.

Para las partidas de miniBridge, la primera fila indicará el número de bazas mínimas y el palo de triunfo, pero no se representará por pantalla. En el caso de las partidas de Bridge la primera fila indicará qué contrato de juego es el más adecuado, y la segunda fila no se representará por pantalla porque las bazas que ha de hacerse la pareja declarante dependen del contrato definido previamente en la subasta.

2.6 Diagramas de casos de uso.

Los diagramas de casos de uso (19) representan el comportamiento de un sistema desde el punto de vista de un sujeto usuario. Estos diagramas representan las funciones que el sistema ofrece al usuario.

Son fáciles de interpretar, mejorando así la comprensión de cara tanto al cliente como al equipo de diseño.

Elementos básicos de los diagramas**:**

- **Actores**: elementos externos que interactúan de algún modo con alguno de las funciones del sistema. Pueden ser humanos o incluso otros sistemas.

- **Sistema**: conjunto de funciones a estudiar.

- **Casos de uso:** es una función o tarea que debe llevar a cabo el sistema. Cada caso de uso se detallará individualmente.

- **Asociaciones:** son las interactuaciones entre los casos de uso, si el actor interactúa con el sistema.

- **Escenario:** es la interacción entre en el sistema y los actores.

El diagrama de casos de usos del proyecto es el siguiente:

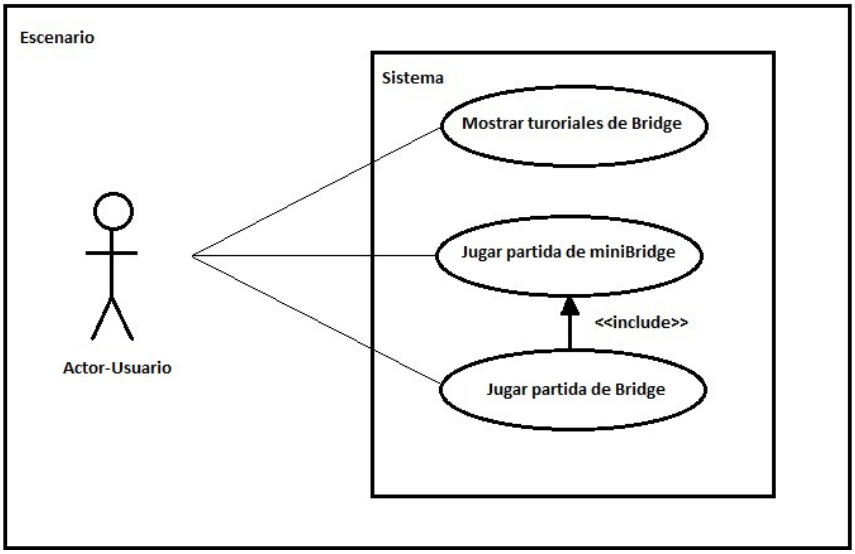


Figura 17. Diagrama de casos de uso

El *software* que se pretende desarrollar tiene tres casos de uso. El primero de ellos, muestra parte de las reglas y los tutoriales de juego para dar a conocer el Bridge y entender los objetivos y características del mismo. Los dos últimos casos de uso interactúan con el usuario jugando diferentes partidas para iniciarse al juego, mejorando así el nivel o como mero entretenimiento. El tipo de partida de Bridge incluye a su vez el tipo de partida miniBridge, durante la segunda fase de juego.

A continuación se procederá a hacer un estudio detallado de cada uno de los tres casos de uso del diagrama anterior con los parámetros especificados y detallados en la tabla 3.

Numeración	Número técnico identificativo
Apelativo	Nombre del caso de uso
Objetivo	Objetivos y funciones al interactuar con el caso de uso
Precondiciones	Estado del sistema para que se lleve a cabo el caso de uso
Descripción	Descripción de las tareas y acciones del caso de uso
Postcondiciones	Condiciones que se formalizan cuando cesa el caso de uso

Escenarios alternativos	Excepciones o alternativas las tarea y funciones normales del caso de uso

Tabla 3. Formato de la descripción de casos de uso

Numeración	**CU.01**
Apelativo	Mostrar tutoriales de iniciación al Bridge.
Objetivo	Familiarizar al usuario con los objetivos y fundamentos principales del bridge.
Precondiciones	Ninguna.
Descripción	**01**. El usuario accede al sistema. **02**. El usuario selecciona un tutorial de juego. **03**. El programa muestra el tutorial. **04**. El usuario cambia las diapositivas. **05**. El usuario termina el tutorial. **06**. El programa vuelve al menú de juego.
Postcondiciones	El sistema permanece en "Menú de juego" para que el usuario acceda a otro tutorial o para que seleccione una partida.
Escenarios alternativos	Ninguno.

Tabla 4. Caso de uso CU.01

Numeración	CU.02
Apelativo	Jugar partida de miniBridge.
Objetivo	Jugar partidas para afianzar y comprender los objetivos, fundamentos y características del miniBridge. Es decir, hacer bazas y entender quién gana cada baza.
Precondiciones	Ninguna.
Descripción	**01**. El usuario accede al sistema. **02**. El usuario selecciona el tipo de partida miniBridge. **03**. El programa accede al tipo de partida miniBridge. **04**. El programa crea todos los objetos de Bridge necesarios para la partida. **05**. El usuario determina cuándo la máquina empieza a jugar el carteo. **06**. El sistema muestra las ayudas y orientaciones de juego. **07**. La máquina juega una carta del jugador en turno. **08**. El usuario juega una carta, en su turno de juego, interactuando con la interfaz. **09.** La máquina juega una carta del jugador en turno. **10**. El usuario juega una carta, en su turno de juego, interactuando con la interfaz. **11**. La máquina evalúa la baza. **12**. El programa vuelve 12 veces más al punto 6 o7 en función del ganador la baza anterior. **13**. El programa calcula la puntuación y muestra la conclusión final. **14**. El usuario escoge entre volver a jugar la mano, jugar otra diferente o finalizar.
Postcondiciones	El sistema modifica los elementos de juego de la mesa como estaban en el inicio para poder jugar otra mano.
Escenarios alternativos	Ninguno.

Tabla 5. Caso de uso CU.02

Numeración	**CU.03**
Apelativo	Jugar partida de Bridge.
Objetivo	Jugar partidas para afianzar y comprender los objetivos, fundamentos y características del Bridge. Es decir, entender y familiarizarse con la mecánica de la subasta así como de las bazas.
Precondiciones	Ninguna.
Descripción	**01**. El usuario accede al sistema. **02**. El usuario selecciona el tipo de partida Bridge. **03**. El programa accede al tipo de partida de Bridge. **04**. El programa crear todos los objetos de Bridge necesarios para la partida. **05**. El usuario determina cuándo empieza la subasta. **06**. El sistema muestra las ayudas y orientaciones de juego. **07**. El jugador en turno da una voz. **08**. El sistema evalúa la voz, y la trayectoria de la subasta. **09**. El usuario da una voz, con el *bidding box*. **10**. El sistema evalúa la voz, y la trayectoria de la subasta n-veces. **11**. Termina la subasta. **12**. El programa modifica la mesa de juego para proceder al carteo. **13**. Ejecutar el **CU.02.05**.
Postcondiciones	El sistema modifica los elementos de juego de la mesa como estaban en el inicio para poder jugar otra mano.
Escenarios alternativos	Ninguno.

Tabla 6. Caso de uso CU.03

2.7 Catálogo de requisitos del proyecto.

En esta sección se van a mostrar todos los requisitos (o tareas) para llevar a cabo correctamente los objetivos de la herramienta de aprendizaje. Para ello se mostrarán tres tablas en la que se reflejarán tres tipos de requisitos, siguiendo los parámetros de la metodología métrica *vol.3* (20).

- **Requisito Funcional (RF)**: referidos o tareas estrictamente funcionales, verificables y objetivas para la correcta realización del proyecto. Son requisitos unívocos y elementales.

- **Requisito No Funcional (RN):** requisitos que es sistema debe cumplir pero que carecen de funcionalidad. Hay varios tipos de requisitos no funcionales. Se destacan los siguientes de cara al proyecto:

    - Requisitos de interfaz
    - Requisitos de seguridad

- **Requisito Inverso (RI):** son los requisitos funcionales y no funcionales que son no requeridos en el proyecto. Se especifican para no llevarse a cabo por error.

Estos requisitos no tienen el mismo grado de importancia o de prioridad. Este aspecto se también se reflejará en las tablas.

2.7.1 Esquema de la tabla de requisitos

Cada uno de los tres tipos de requisitos anteriormente citados, se presentaran en una tabla propia con el formato esquemático de la Tabla 7. Cabe mencionar que estos requisitos ha sido resultado de diferentes versiones, incluyéndose únicamente en la tabla las versiones finales.

Identificador	Apelativo	Prioridad	Descripción
Nº técnico identificativo	**Descripción breve**	**Prioridad de implementación respecto a otras tareas**	**Explicación detallada unívoca**

Tabla 7. Formato de especificación de requisitos

2.7.2 Requisitos Funcionales.

Id.	Apelativo	Prioridad	Descripción
RF.01	Orientado a sistemas operativos comunes	**Alta**	El sistema podrá ejecutarse en los sistemas operativos Windows, Mac y Linux.
RF.04	Crear una vista que contenga el menú de juego	**Alta**	El sistema debe facilitar el acceso a tutoriales o a partidas indistintamente.
RF.07	Crear una vista con la modalidad de partida de miniBridge	**Alta**	El sistema permitirá jugar una partida de miniBridge para que el usuario se familiarice con las bazas.
RF.08	Crear una vista con la modalidad de partida de Bridge	**Alta**	El sistema permitirá jugar una partida de Bridge para que el usuario aprenda a jugar la subasta y el carteo
RF.09	Crear un único nivel de juego máquina en el carteo	**Alta**	El sistema deberá disponer de lógica interna para decidir qué cartas jugará durante la partida en función de las acciones del usuario y las manos. (Nivel intermedio).
RF.10	Crear un único nivel de juego máquina en la subasta	**Alta**	El sistema deberá disponer de lógica interna para decidir qué voces subastará durante la partida en función de las acciones del usuario y las manos. (Nivel Básico)
RF.11	Leer las ayudas de un documento de texto*(txt)*	**Alta**	El sistema debe leer las ayudas de un documento de texto para poder reflejarlas por pantalla.
RF.12	Reflejar ayudas escritas por pantalla sobre cómo jugar las manos	**Alta**	El sistema debe facilitar al usuario una orientación sobre el juego.
RF.13	Indicar por texto en la pantalla los cambios de fase durante la partida.	**Alta**	El sistema debe facilitar la comprensión de los cambios de fase durante la partida para que el usuario este orientado en todo momento.
RF.14	Analizar el ganador de cada baza	**Alta**	El sistema debe ser capaz de calcular el ganador de cada baza en función de la carta inicial de la baza y el palo de triunfo.

Id.	Apelativo	Prioridad	Descripción
RF.15	Reasignar turnos una vez finalizada la baza	**Alta**	El sistema debe ser capaz reasignar los turnos para la siguiente baza una vez terminada cada una.
RF.16	Analizar el ganador de la Subasta	**Alta**	El sistema, una vez terminada la subasta, debe analizar el jugador ganador de la esta.
RF.17	Asignar turnos en la subasta	**Alta**	Antes de comenzar la subasta, el programa asignará los turnos para la subasta en función del *dealer* de la mano.
RF.18	Reflejar por la vista de partida el número de mano	**Baja**	El sistema representará por pantalla el número de mano que se está jugando.
RF.19	Reflejar en la vista el contrato de juego	**Media**	El sistema representará por pantalla el contrato de juego que se está jugando, en la modalidad de partida Bridge
RF.20	Representar por la vista el marcador de bazas de cada pareja	**Alta**	El sistema debe incorporar en la pantalla un marcador de bazas de cada pareja para orientar al jugador cuántas bazas lleva ganadas y cuantas perdidas.
RF.21	Evitar renuncios	**Alta**	El sistema debe evitar en todo momento que cualquier jugador cometa un renuncio.
RF.23	Controlar los turnos durante el carteo	**Alta**	El sistema debe controlar que ningún jugador se anticipe a otro durante el carteo.
RF.24	Controlar los turnos durante la subasta	**Alta**	El sistema debe controlar que ningún jugador se anticipe a otro durante la subasta.
RF.25	Controlar que no se rebaje el nivel de la subasta	**Alta**	El sistema debe controlar durante la subasta que no se rebaje el nivel la misma.
RF.26	Interpretar el contrato de la subasta final e incorporarlo en el carteo	**Alta**	Una vez terminada la subasta, el sistema debe interpretar el contrato final de la subasta y traducirlo al carteo en forma de número de bazas y palo de triunfo.

Id.	Apelativo	Prioridad	Descripción
RF.29	Representar las cartas jugadas de la baza	**Alta**	Cuando un jugador juegue una carta de su mano, el sistema debe representar dicha carta sobre la mesa de juego.
RF.30	Representar sobre la mesa las voces dadas por los jugadores.	**Alta**	Cuando un jugador de una voz del *bidding box*, el sistema debe representar dicha voz sobre la mesa de juego.
RF.31	Retirar del juego y de las manos de los cuatro jugadores las cartas una vez sean jugadas	**Alta**	Una vez se haya jugado una carta, el sistema debe retirarla de la mano del propietario.
RF.32	Retirar de la mesa las cartas de la baza	**Alta**	Una vez terminada cada baza, cuando se proceda a jugar la siguiente, el sistema debe retirar las cartas de la baza en la mesa.
RF.33	Modificar la vista entre la subasta y el carteo	**Alta**	Cada vez que se termine una fase del juego o se acabe la mano, el programa debe reiniciar todas las variables para la siguiente fase o mano.
RF.36	Modalidad de juego sin ayudas	**Baja**	El sistema debe poder dar a elegir al usuario si quiere ayudas o no. "Las ayudas estarán por defecto".
RF.37	Dar a elegir empezar carteo	**Media**	El sistema debe permitir al usuario cuándo quiere empezar el carteo.
RF.38	Dar a elegir empezar subasta	**Media**	El sistema debe permitir al usuario cuándo quiere empezar la subasta.
RF.39	Deshacer bazas y voces	**Alta**	El sistema debe poder dar la opción al usuario de retirar de la mesa una carta o voz jugada y devolverla a la mano o al *bidding box*. Además deberá retirar las cartas o voces jugadas por los jugadores máquina.
RF.40	Dar la posibilidad de volver a la vista menú de juego	**Alta**	El sistema debe dar la posibilidad de volver al menú de juego para cambiar de caso de uso.

Id.	Apelativo	Prioridad	Descripción
RF.41	Dar a elegir una nueva mano	**Media**	El sistema debe dar la opción al jugador de elegir una nueva mano.
RF.42	Reflejar una imagen con las cartas de la última baza jugada.	**Media**	El sistema representará en una "subventana" una imagen con las cartas de la baza anterior.
RF.43	Dar la opción de repetir la mano	**Alta**	El sistema, si así lo solicita el usuario, repartirá otra vez la mano desde el principio.
RF.46	Representar una evaluación al finalizar cada fase de juego	**Alta**	El sistema debe analizar si el juego del usuario ha sido correcto al finalizar la subasta y el carteo.
RF.47	Asignar puntuaciones de Bridge	**Media**	El sistema debe ser capaz de calcular la puntuación de Bridge correcta una vez finalizada cada mano.
RF.48	Representar por la vista las voces de doblo y redoblo, pero no incluirlas en el juego.	**Alta**	El sistema representará por pantalla las voces de doblo y redoblo, pero no se podrán utilizar para la subasta por ser voces de nivel más avanzado.
RF.49	Lectura de manos en formato *CSV* generador por el programa *Dealmaster Pro*.	**Alta**	El sistema debe ser capaz de leer e interpretar ficheros en formato *CSV* provenientes del programa *DealMaster Pro* para poder representar la mano reflejada en el fichero sobre la vista. De este modo el cliente puede incorporar un número ilimitado de manos. A su vez cualquier profesional puede incorporar las manos que el crea convenientes para sus alumnos.
RF.50	Dar la posibilidad de abandonar la mano y jugar otra en cualquier momento	**Media**	El sistema debe dar la posibilidad de terminar la mano en cualquier momento que el usuario lo considere oportuno.

Tabla 8. Requisitos Funcionales

2.7.3 Requisitos no Funcionales
- De interfaz

Id.	Apelativo	Prioridad	Descripción
RN.01	Idioma castellano	**Alta**	El sistema debe representar todos los títulos, tutoriales y ayudas en castellano.
RN.02	Incluir un único tipo de baraja, para todos los tipos de partidas y tutoriales	**Alta**	El sistema debe tener coherencia entre barajas.
RN.04	Símbolos de la baraja Inglesa	Media	El sistema pretende implementar una estética moderna.
RN.05	Incluir una baraja con picas y tréboles negros, y corazones y diamantes rojos	**Media**	El sistema incluirá como baraja de juego con los colores clásicos de cada palo, es decir, rojo y negro.
RN.06	Solo dos números por cada carta de la baraja	**Media**	El sistema debe implementar una baraja que solo tenga dos números, o letras, por carta y no uno ni cuatro, porque no hace falta una baraja compatible para zurdos.
RN.07	Baraja de bridge.	**Alta**	El sistema debe implementar una baraja que tenga una proporción de carta de bridge y no una carta de tamaño póker.
RN.08	Color base y principal de la pantalla : (153, 153, 255)	**Alta**	El sistema debe incorporar este color porque es el color corporativo de la empresa del cliente.
RN.09	Gama cromática acorde entre los colores principales de la interfaz	**Alta**	El sistema deberá evitar, en la medida de lo posible, el cansancio a la vista y crear una interfaz sencilla, es decir, con colores compatibles.
RN.10	Disposición de la mano del muerto separada por palos y en columnas	**Alta**	El sistema debe representar la mano del muerto separada por palos y en columnas para representar fielmente el estilo de juego y darle un toque moderno a la pantalla.

Id.	Apelativo	Prioridad	Descripción
RN.11	El usuario siempre será el jugador Sur y el declarante	**Alta**	El sistema ubicará siempre al usuario en la posición Sur, para así, facilitar la lectura y comprensión de los sucesos durante la partida.
RN.12	Representar las voces **reales** de la subasta en la mesa	**Alta**	Según se vayan jugando voces durante la subasta, el sistema debe ir representándolas sobre la mesa en el lado del jugador que haya nombrado dicha voz. Estas tienen que estar representadas realmente y no simbólicamente.
RN.13	Representar *bidding box* real	**Alta**	El sistema debe representar el *bidding box* debe ser real y no esquemático.
RN.14	Incorporar veinte de manos	**Media**	El sistema debe incorporar veinte manos diferentes y de un nivel de iniciación.
RN.15	Tutorial de iniciación al miniBridge,	**Alta**	El sistema ofrecerá un tutorial de miniBridge que sea una secuencia de imágenes, para dar a conocer los objetivos **básicos** del juego
RN.17	Tutorial de iniciación al Bridge	**Alta**	El sistema ofrecerá un tutorial de Bridge que sea una secuencia de imágenes, para dar a conocer **todos** los del juego

Tabla 9. Requisitos No Funcionales de interfaz

- De Seguridad

Id.	Apelativo	Prioridad	Descripción
RS.04	Confirmar el deseo de cambiar de mano.	**Media**	El sistema debe requerir del usuario la confirmación de deseo de cambiar a otra mano a elección.
RS.06	Confirmar el deseo de salir a la pantalla menú de juego	**Alta**	El sistema debe requerir del usuario la confirmación de deseo de abandono de la partida.

Tabla 10. Requisitos No Funcionales de seguridad

2.7.4 Requisitos inversos

Id.	Apelativo	Descripción
RI.01	No incluir más de un nivel de juego máquina	No son las competencias del proyecto
RI.02	No repartir partidas con manos aleatorias	El sistema no someterá al usuario a partidas complicadas desde el inicio.
RI.03	No incorporar cartas de Póker	El sistema no permitirá que se pueda asociar visualmente el Bridge con el Póker
RI.04	Sin contraseñas de acceso a las vistas	El sistema debe permitir que el usuario decida por sí mismo los casos de uso que crea convenientes en cada instante.

Tabla 11. Requisitos Inversos

2.8 Arquitectura preliminar

Para abordar el problema de diseño de los componentes del *software*, nos ayudaremos de un patrón de diseño. Este patrón es una descripción de clases y objetos comunicados entre sí adaptada para resolver un problema general de diseño en un contexto particular. En este proyecto, se analizará el patrón de diseño con diagramas **Modelo Vista Controlador** (*MVC*).

Los diagramas *MVC* (25), consisten en la separación de la aplicación del *software* en las tres componentes principales que se detallan a continuación. El objetivo principal es la reutilización del código ya implementado. Comúnmente estos diagramas son usados en las interfaces de usuario para solucionar problemas de actualización de la aplicación cuando se modifican las variables de la misma.

A continuación se describen los tres componentes principales del diagrama MVC, Figura 18.

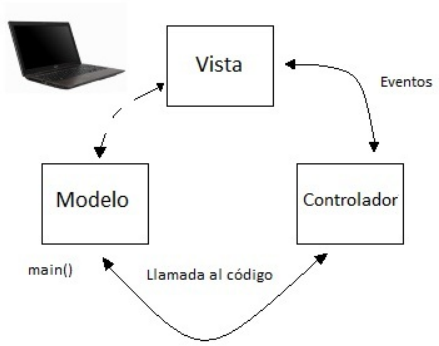


Figura 18. Diagrama MVC

- **Modelo**: es la representación específica del dominio de la información. Los códigos del domino dan significado a los datos.

- **Vista**: es una interfaz de usuario donde se presenta el Modelo en un formato adecuado para interactuar con el usuario.

- **Controlador:** hace de conexión entre la Vista y el Modelo, pudiendo provocar cambios en cualquiera de ellos.

2.9 Plan de pruebas del software

En este apartado se procede a hacer un análisis de las pruebas que se realizarán una vez finalizadas las fases de implementación de cada uno de los componentes del *software*. Estas pruebas tienen como objetivo verificar si se cumplen los requisitos funcionales detallados en la sección 2.7.2. Las pruebas detalladas son un resumen de las múltiples pruebas que se llevarán a cabo.

Consideraremos superadas las pruebas cuando se cumplan el 100% de los requisitos funcionales. En caso de que alguna de las pruebas fallara, se revisará la implementación. Si la prueba continuase fallando, se procederá a modificar los diseños e implementaciones pertinentes.

Una vez finalizadas las pruebas satisfactoriamente, se puede dar por concluido el proceso de desarrollo del *software*.

2.9.1 Formato de tabla del plan de pruebas

En la siguiente tabla se representa el formato de especificación de cada una de las pruebas que se llevarán a cabo una vez finalizada la implementación de cada uno de los componentes (sección 2.8).

Identificador	**Numero técnico identificativo**
Nombre	**Descripción breve**
Objetivos	**Objetivos de la prueba**
Requisitos en estudio	**Requisitos sometidos a estudio**
Proceso	**Pasos a realizar durante la prueba**
Resultado	**Resultado esperado al concluir la prueba**

Tabla 12. Formato de las pruebas del software

2.9.2 Pruebas del software

Como se ha mencionado anteriormente, en las pruebas que se llevarán a cabo, se evaluarán los requisitos funcionales del sistema (sección 2.7). Por ello se hará una prueba diferente para evaluar las interfaces, la fase y lógica de subasta, la fase y lógica del carteo, la fase de ayudas, las conclusiones finales y las opciones de juego.

Identificador	**PS.01**
Nombre	Representación de los elementos en las vistas
Objetivos	Comprobar que las vistas se crean correctamente y los elementos de las mismas no modifican su ubicación durante el transcurso de los casos de uso.
Requisitos en estudio	RF.01/RF.04/RF.07/RF.08/RF.18/RF.19/RF.20/RF.33/RF.48/RF.49

Proceso	**01**. Iniciar el programa. **02**. Comprobar que la pantalla menú se representa correctamente. **03**. Seleccionar los tutoriales. **04**. Comprobar que la vista se ha creado correctamente. **05**. Cerciorarse de que las imágenes y los elementos de tutorial se representan correctamente. **06**. Comprobar que el cambio de imagen/página es correcto. **07**. Volver a la pantalla menú y comprobar que se representa correctamente. **08**. Seleccionar partida miniBridge. **09**. Comprobar que la carta de Oeste se representa correctamente. **10**. Jugar una carta del muerto. **11**. Comprobar que la carta desaparece de la mano y se representa en la mesa correctamente. **12**. Comprobar que las cartas del muerto se ajustan una vez retirada la carta jugada. **13**. Repetir puntos 9, 10, 11 y 12 para Este y Sur. **14**. Terminar de jugar la mano, comprobando en cada baza que el recuento de las mismas se representa adecuadamente. **15**. Una vez terminada la mano, comprobar que los dorsos de las cartas se representan por la pantalla. **16**. Volver a la pantalla menú y comprobar que se representa correctamente. **17**. Seleccionar partida de Bridge. **18**. Comprobar que todos los elementos están dispuestos y creados adecuadamente. **19**. Comprobar que las voces de la subasta se representan correctamente en la mesa y unas encima de las otras. **20**. Una vez terminada la subasta, comprobar que la mesa y la pantalla se modifican correctamente para el carteo y que desaparecen las voces de la mesa. **21**. Comprobar que el contrato se muestra sobre la pantalla. **22**. Repetir para el carteo de la partida de Bridge los puntos [6-15].
Resultado	Que se hayan representado todos los elementos correctamente y no se hayan movido durante los casos de uso.

Tabla 13. PS.01 Representación de las vistas

Identificador	**PS.02**
Nombre	Proceso de subasta
Objetivos	Comprobar que el proceso de la fase de subasta es correcto.
Requisitos en estudio	RF.10/RF.16/RF.17/RF.19/RF.24/RF.25/RF.30/RF.38/RF.39
Proceso	**01**. Iniciar el programa. **02**. Seleccionar partida de Bridge. **03**. Seleccionar empezar. **04**. Comprobar que la máquina interpreta bien el *dealer* y éste da una voz. **05**. Comprobar si el *bidding box* se modifica correctamente. **06**. Comprobar que la secuencia de turnos se lleva a cabo correctamente en función de las reglas. **07**. Comprobar que las voces que va dando la máquina cumplen las reglas y no rebajan el nivel de la subasta. **08**. Seleccionar una voz en el turno del usuario mediante el *bidding box*. **09**. Comprobar que el sistema interpreta correctamente la voz dada y la representa en la mesa de juego. **10**. Seleccionar deshacer. **11**. Comprobar que las voces pertinentes han sido retiradas de la mesa y el *bidding box* ha modificado las voces que se pueden dar. **12**. Analizar si la máquina sigue llevando a cabo la subasta correctamente una vez pulsado el deshacer. **13**. Finalizar la subasta. **14**. Analizar si la subasta ha sido correcta, por parte de los jugadores máquina, en función de la técnica de juego implementada.
Resultado	Que todas las reglas de la subasta hayan sido ejecutadas correctamente así como que el programa haya ejecutado correctamente la opción de deshacer. Además el nivel de juego de los jugadores máquina ha debido ser apropiado durante toda la subasta.

Tabla 14. PS.02 Proceso de subasta

Identificador	**PS.03**
Nombre	Proceso de carteo
Objetivos	Comprobar que el proceso de la fase de carteo es correcto.
Requisitos en estudio	RF.07/RF.09/RF.14/RF.15/RF.21/RF.23/ RF.26/RF.29/RF.31/RF.32/RF.33/RF.37/RF.39/RF.40/RF.42
Proceso	**01**. Iniciar el programa. **02**. Seleccionar partida de miniBridge. **03**. Seleccionar empezar. **04**. Comprobar que Oeste juega una carta. **05**. Comprobar si la máquina bloquea las cartas del muerto que no pueden jugarse por la regla de asistir al palo. **06**. Comprobar que las cartas de Sur están bloqueadas. **07**. Comprobar que la carta de Este también cumple la regla de asistir al palo. **08**. Comprobar que la máquina ha bloqueado las cartas de norte y ha desbloqueado las cartas de sur en función de la regla asistir al palo. **09**. Comprobar que la máquina evalúa el ganador de la baza perfectamente y asigna los turnos para la siguiente. **10**. Seleccionar deshacer. **11**. Comprobar que las cartas pertinentes vuelven a las manos de sus propietarios y son retiradas de la mesa. **12**. Analizar si la máquina sigue llevando a cabo los turnos de carteo correctamente. **13**. Finalizar el carteo. **14**. Una vez finalizado el carteo analizar si los jugadores máquina han llevado a cabo una defensa correcta en función de las técnicas de juego implementadas. ***** Esta prueba es exacta para el carteo del Bridge, pero con la excepción de que hay que comprobar si la máquina identifica correctamente el palo de triunfo del contrato.
Resultado	Que todas las reglas de juego hayan sido adecuadas y el programa ejecute correctamente la opción de deshacer. Además el nivel de juego de la máquina ha debido ser apropiado

Tabla 15. PS.03 Proceso de carteo

Identificador	**PS.04**
Nombre	Proceso de ayudas
Objetivos	Comprobar que el proceso de lectura y representación de ayudas es correcta.
Requisitos en estudio	RF.11/RF.12/ RF.13/RF.36
Proceso	**01**. Iniciar el programa. **02**. Seleccionar partida de miniBridge. **03**. Seleccionar empezar. **04**. Proceder al carteo. **05**. Comprobar que las ayudas se reflejan correctamente. **06**. Analizar las ayudas y verificar que corresponden a la mano que se está jugando. **07**. Volver al menú. **08**. Seleccionar partida de Bridge. **09**. Seleccionar empezar. **10**. Comprobar que la máquina trasmite las ayudas durante la subasta. **11**. Analizar si las ayudas transmitidas son correctas.
Resultado	Que todas las ayudas de juego hayan sido obtenidas y representadas correctamente por pantalla.

Tabla 16. PS.04 Proceso de ayudas

Identificador	**PS.05**
Nombre	Finalización de las manos
Objetivos	Comprobar que se representa correctamente la conclusión al final de cada mano.
Requisitos en estudio	RF.11/RF.12/ RF.13/RF.36
Proceso	**01**. Iniciar el programa. **02**. Seleccionar partida de Bridge. **03**. Iniciar subasta **04**. Finalizar la subasta. **05**. Proceder al carteo. **06**. Finalizar el carteo. **07**. Comprobar que una vez finalizada la mano, el programa representa por pantalla una conclusión final. **08**. Analizar la puntuación asignada y comprobar que es correcta en función del contrato y el recuento final de bazas.

Resultado	La conclusión final debe ser correcta.

Tabla 17. PS.05 Finalización de las manos

Identificador	**PS.06**
Nombre	Opciones de cambio de juego
Objetivos	Comprobar que las opciones de cambio de mano y la salida de partida se llevan a cabo correctamente.
Requisitos en estudio	RF.40/RF.41/ RF.43/RF.50
Proceso	**01**. Iniciar el programa. **02**. Seleccionar partida de miniBridge. **03**. Seleccionar un cambio de mano. **04**. Comprobar que el cambio de mano es correcto. **05**. Comprobar que todos los elementos se reinician al estado inicial. **06**. Salir de la partida. **07**. Seleccionar partida de Bridge. **08**. Proceder a la subasta. **09**. Seleccionar cambio de mano. **10**. Comprobar que todos los elementos se reinician al estado inicial. **11**. Proceder a la subasta de la nueva mano **12**. Finalizar subasta. **13**. Proceder al carteo. **14**. Seleccionar cambio de mano. **15**. Comprobar que todos los elementos se reinician al estado inicial.
Resultado	El cambio de mano ha tenido que modificar todos los elementos de juego correctamente.

Tabla 18. PS.06 Opciones cambio de juego

2.10 Análisis del juego

En este apartado se incluye principalmente parte del análisis llevado a cabo sobre cómo se debe proceder a un nivel de carteo intermedio y a un nivel de subasta básico. No se han incluido en su totalidad para salvaguardar los derechos del autor sobre dicho análisis. Además se presentaran diversas alternativas para la implementación de esta lógica de juego. Al final del documento se ha incluido el análisis del cálculo completo de las puntuaciones. La *máquina1* representa al jugador Oeste y la *máquina2* representa al jugador Este.

Recuérdense los tecnicismos especificados en la sección 1.4, para la comprensión correcta de este apartado.

2.10.1 Técnicas de programación aplicables.

Para la implementación de la lógica de juego de los jugadores máquina de este *software* se pueden utilizar diferentes tipos de programación.

Uno de estos tipos de programación son las técnicas de razonamiento automático, como es el ejemplo de las *Redes de Neuronas*. Gracias a esta técnica se podrían predecir las cartas que se jugaran posteriormente. Otro tipo de programación asequible es el razonamiento con incertidumbre mediante *Lógica Difusa* ofreciendo técnicas de predicción parecidas a las anteriores.

El último tipo de programación que se baraja para la implementación de tal lógica es mediante la programación tradicional son sentencias *For* e *If*. Esta última es la más sencilla de implementar pero también la menos precisa.

2.10.2 Lógica de juego del carteo

Se detalla la decisión que tomará el jugador *máquina1* para empezar el carteo. El proceso a seguir para deducir la carta más apropiada, para la salida inicial al carteo, dependerá de si el contrato es a "palo" o a "ST". Para la no dar lugar a error, se ha procedido a eliminar la secuencia numérica para los títulos.

1. Salida a ST.

1.1 Si *máquina2* no ha dado voz

- Tendrá que jugar del palo más largo de la mano, siempre y cuando no sea el palo largo del declarante. En este último

caso saldrá del siguiente palo más largo si no ha sido nombrado por los oponentes.

- Una vez analizado qué palo es el adecuado, se estudiará si puede jugar una de esas cartas. Por el contrario se pasará a analizar otro palo.
    - Si tiene una secuencia de honores, mínimo tres, hay que salir del honor de mayor rango (para afianzar los demás).
      Ejemplo KQJ63. Saldrá del K.

    - Si no se tiene secuencia, pero tienen tres honores siendo los dos honores mayores consecutivos y el tercero es el siguiente al que falta, hay que salir del honor mayor (semisecuencia).
      Ej. A K J 3 2. Saldrá de As aunque falte la Q.

- Si tiene una secuencia interna, saldrá de la carta mayor de la secuencia interna.
  Ej. A Q J 10 → saldrá de la Q

- Si en el palo más largo no hay honores o solo uno, saldrá de la segunda por abajo.
  Ej. 10 9 5 3 2. Saldrá del 3.
  8 7 3 2. Saldrá del 3.

- Si este palo no tiene honores pero sí secuencia o semisecuencia, se llevará a cabo el mismo procedimiento que en la primera condición. Si hay dos secuencias o semisecuencias se saldrá de la mayor.

- Si en el palo hay dos honores consecutivos, o un único honor, saldrá de la cuarta carta del palo contando por arriba.

- Si se han nombrado todos saldrá de un palo de tres cartas sin honores, concretamente de la carta más alta.

1.2 Si la *máquina2* ha nombrado un palo.

- Si la *máquina1* tiene secuencia en este palo saldrá de la carta mayor.

- Si no se tiene secuencia procederá a los siguiente:

- Si tiene una de su palo jugará esa.
- Si tiene dos cartas en ese palo saldrá de la mayor.
- Si tiene 3 cartas de la más pequeña.
- Si tiene 4 de la segunda por abajo.
- Si tiene 5 se actúa como en apartado (1.1)
- Si no tiene ninguna carta en ese palo saldremos como si nuestro compañero no hubiera dicho nada, es decir, el apartado anterior (1.1).

Una vez ha salido la *máquina1*, se extenderá el muerto. Tendrá el turno el jugador muerto que lo controla el usuario a voluntad. Una vez el usuario juegue una carta del muerto, el turno es de la *máquina2*. Esta máquina tendrá que analizar si tiene cartas del palo de salida en su propia mano.

1.3 Si *máquina2* tiene cartas del palo salida de la *máquina1*.

- En el caso de que la máquina tenga únicamente dos cartas, jugará la mayor de todas.

- Si tiene 3 cartas o más;

  Si la carta de la máquina es un honor, jugará carta la más pequeña. Pero si el muerto ha superado la carta que ha puesto la *máquina1*, la *máquina2* debe poner la carta siguiente superior en rango. Si no puede superar el rango jugará la más pequeña.

1.3.1 Si no tiene cartas en ese palo

Jugará la carta más pequeña del palo que mejor y más cartas tengan. Para que la otra *máquina1* identifique el palo bueno de la *máquina2.*

1.4 Si el jugador *máquina1* ha ganado la baza de salida.

Jugará siguiendo el mismo procedimiento hasta que no le queden más cartas. Pero si ha salido de secuencia en vez de ser de tres cartas, la secuencia es de 2 cartas. Pero si el compañero ha puesto una carta alta, parar y jugar como si se saliera bajo impases.

2.10.3 Lógica de juego de la subasta

Según el análisis, la subasta es más sistemática que el carteo. Cada una de los jugadores máquina contará la fuerza de su mano según el siguiente baremo de puntos:

Rango	Puntos de honor
A	4
K	3
Q	2
J	1

Tabla 19. Puntos de honor

Cada uno de los jugadores máquina calculará la suma total de todos los puntos que le otorga cada carta de honor que tenga en posesión. Una vez finalizado el cálculo, las máquinas procederán al siguiente análisis:

- **Si ningún jugador ha dado una voz de nivel:**
  - Si la suma de sus puntos es menor de 12 puntos dará la voz de PASO.
  - Si la suma de sus puntos está comprendida entre 12 y 20 puntos, ambos incluidos, dará una voz a nivel de uno de un palo mayor si tiene mínimo cinco cartas en ese palo. Si no es así y tiene entre 16 y 18 puntos y una mano regular dará la voz de 1ST. Si no cumple ninguna de estas dos condiciones dará una voz a nivel de uno del palo menor con mayor número de cartas.
- **Si un jugador ha dado una voz de nivel.**
  - Si es del compañero:
    - Evaluará el palo de la voz del compañero. Si es un palo mayor y tiene tres o más cartas de ese palo, en función de sus puntos dará la voz de nivel correspondiente de ese mismo palo. Si tiene menos de 5 puntos dirá PASO.

- Con más de 6 puntos de honor, si la voz del compañero es a palo menor, a nivel de uno, dará los palos mayores de cuatro o más cartas. Si no tiene ningún palo mayor de cuatro o más cartas dirá la una voz de nivel de ST en función de sus puntos y de si su mano es regular. Si la mano no es regular y tiene más de 10 puntos dirá el palo menor con más cartas.

    - 1ST entre 6 y 9 puntos.
    - 2ST entre 10 y 12 puntos
    - 3ST con 13 y 18 puntos.

2.10.4 Puntuación

Para realización del cálculo correcto de puntuaciones se ha utilizado la siguiente referencia, sin incluir la vulnerabilidad ni las voces de doblo y redoblo. En primer lugar se analizan tres aspectos:

1. Número de bazas logrado en el carteo por la pareja declarante.
2. Contrato definido en la subasta por la pareja declarante
3. En este programa no se calculan las vulnerabilidades ni las voces de doblo y redoblo.

El contrato establece el número de bazas que debe cumplir la pareja declarante durante el contrato, definido por esta suma:

Nivel Contrato+ 6 =Nº mínimo de bazas.

Si el nivel del contrato es dos, las bazas que tendrá que ganar serán ocho. Si por ejemplo el nivel del contrato es siete, ha de ganar trece bazas, es decir, todas. Por ello, el número de mayor nivel que se puede alcanzar en la subasta es siete.

Si la pareja declarante ha conseguido alcanzar el número acordado de bazas, se aplicarán unas puntaciones en función de estos tres factores:

1. Tipo de contrato (Parcial, Manga o Slam).
2. Palo del contrato.
3. Número de bazas extra.

Estos tres factores se incluyen en las siguientes tablas.

- **Contrato parcial**

Puntuación por defecto	Triunfo	Puntos por baza a partir de la sexta
50 +	♦ o ♣	20
	♠ o ♥	30
	ST Primera baza Siguientes bazas	40 30

Tabla 20. Puntuación contrato parcial

- **Manga**

Puntuación por Manga	Triunfo	Puntos por baza a partir de la sexta
300 +	♦ o ♣	20
	♠ o ♥	30
	ST Primera baza Siguientes bazas	40 30

Tabla 21. Puntuación contrato manga

- **Slams**

Puntuación por incluir Manga	Puntuación por Slam	Triunfo	Puntos por baza a partir de la sexta
300+	***Nivel 6.*** **500+**	♦ o ♣	20
		♠ o ♥	30
	Nivel 7. **1000+**	**ST** Primera baza Siguientes bazas	40 30

Tabla 22. Puntuación contrato slam

- **Multas**

Cada baza por debajo de las prometidas, es una multa. La pareja defensora recibirá 50 puntos por multa, y la pareja declarante no recibirá ningún punto.

3. DISEÑO

A lo largo de este capítulo se procederá a documentar el proceso de diseño que se llevará a cabo a partir de los objetivos, el análisis de tecnologías, los casos de uso, los requisitos y el análisis de la arquitectura preliminar que se detallaron en el capítulo anterior. Además, tendrá como objetivo facilitar y guiar todo el proceso de implementación detallado en el siguiente capítulo.

Para representar y describir correctamente los componentes que formarán parte del diseño se utilizarán diagramas de Lenguaje Unificado Modelado ***UML***. Este es un leguaje específico para representar, detallar y documentar, en nuestro caso, el *software*. Este lenguaje fue creado por y para universalizar un lenguaje para documentar todo tipo de proyectos de industria (21).

En primer lugar se presenta la selección de tecnologías con las que se diseñará e implementará el sistema (sección 3.1). Posteriormente se representa el diagrama de componentes que formarán parte del *software* (sección 3.2). A continuación se detallarán los planos arquitecturales del componente Vista así como los diagramas de clases de cada uno de los componentes del sistema (sección 3.3). Por último se incorporan los diagramas de secuencia de los casos de uso(sección 3.4).

3.1 Selección de tecnologías

En esta sección se detallarán las tecnologías que se utilizarán para llevar a cabo el diseño de los componentes de nuestro *software* (sección 3.2).

3.1.1 Lenguaje de programación

Para el desarrollo general del *software* se ha utilizado el conocido lenguaje de programación ***Java*.**

Esta elección se justifica en su característica principal, un lenguaje abierto e independiente del sistema operativo y orientado a la creación de objetos. El *software* que se desarrollará con este proyecto contiene objetos tales como cartas, voces de subasta, jugadores, un controlador de las partidas e interfaces. Para la creación de estos objetos de una manera algorítmica, *Java* se convierte en una herramienta muy útil, rápida y eficaz.

Otros dos de los principales motivos para la elección de este lenguaje son la característica de Java con el colector de basuras automático (liberación de memoria) y el *API* (26) que facilita clases ya creadas y de gran utilidad. Estas ventajas dotan de

sencillez y fiabilidad al programa, ahorrando a su vez tiempo durante la etapa de implementación.

3.1.2 Entorno de programación.

El entorno de programación que se utilizará para escribir y ejecutar el lenguaje *Java* será *NetBeans (28)*. Concretamente la versión *NetBeans 6.9.1,* de la que se dispone en el laboratorio. Por estudios y proyectos anteriores, este entorno garantiza un funcionamiento acorde a las exigencias y competencias del *software*.

3.1.3 Programación de la lógica de juego.

Para desarrollar la lógica de juego, que tendrán que implementar los jugadores máquina, se ha elegido entre los tipos de técnicas detalladas en la sección 2.10.1, la técnica de programación tradicional mediante sentencias *For* e *If*. Con esta programación se espera que la máquina alcance el nivel de juego acordado en los requisitos.

3.1.4 Diseño de los colores de la pantalla de las partidas.

En base a cumplir el requisito RN.07, para el correcto diseño de la gama cromática de los colores de la vista partida se utilizará la herramienta *Kuler* de *Adobe* (23), disponible gratuitamente en internet. Esta herramienta es usada por diseñadores de todo el mundo para la creación de páginas web, *banners* publicitarios y todo tipo de carteles. *Kuler*, en nuestro caso, nos permitirá crear gamas cromáticas de colores a partir de un color base "(153, 153, 255)" especificado en el requisito RN.06.

3.1.5 Diseño de las imágenes de los tutoriales.

Para la implementación de las imágenes de los tutoriales se recomienda utilizar el programa *Adobe Photoshop CS3* (22) ya que con esta herramienta se obtienen imágenes de gran calidad. Además permite elegir la posición exacta, mediante capas, de cada uno de los elementos de dichas imágenes, facilitando así la etapa de implementación.

3.2 Diagrama de componentes y de despliegue

En este apartado se presenta el diagrama de los componentes que formarán parte del proyecto. Este diagrama (Figura 19) servirá como punto de referencia para, posteriormente, explicar cada una de las partes principales de las que se compone el sistema y sus funciones.

El diagrama de componentes incluye un componente global que es el *hardware o* el sistema operativo en el que se implementará el *software.*

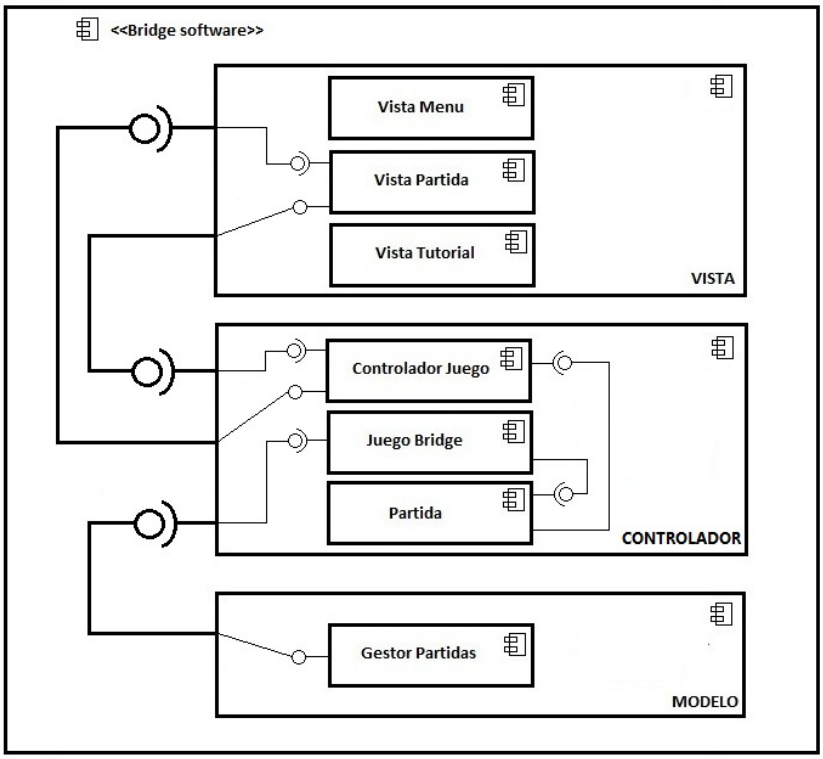


Figura 19. Diagrama de componentes del Software

Como se analizó y estudió en el capítulo de análisis, concretamente en la sección 2.3, nuestro *software* consta de tres grandes componentes, el Modelo, la Vista y el Controlador.

El componente Vista lo forman tres subcomponentes principales, *VistaMenú*, *VistaPartida* y *VistaTutorial*. Desde *VistaMenú* se puede elegir con qué caso de uso desea interactuar el usuario. En función del caso de uso elegido el *software* implementará *VistaPartida* o *VistaTutorial*. Este componente será la interfaz gráfica de usuario.

El siguiente componente que aparece en el diagrama es el Controlador. Este componente se compone de tres subcomponentes principales, *ControladorJuego*,

JuegoBridge y *Partida*. El subcomponente *ControladorJuego* se encarga de controlar y dirigir toda la lógica del programa durante los casos de uso CU.02 y CU.03 (sección 2.5). El subcomponente *JuegoBridge* es el encargado de crear toda la lógica del juego, es decir, barajas, *bidding box*, manos y las puntuaciones en función de los datos del componente Modelo. El subcomponente *Partida* es el encargado de generar los jugadores, otorgarles una posición en la mesa, asignar las manos a sus jugadores propietarios y ordenarlas. Para puntualizar, el subcomponente *JuegoBridge* se encargará de crear los elementos de cualquier partida de Bridge, y el subcomponente *Partida* se encargará de generar, en función de los elementos creados en *JuegoBridge*, una partida específica.

Por último, el componente Modelo, es el encargado de conectar con la base de datos. El subcomponente *GestorPartidas* lleva a cabo la lectura e interpretación de los ficheros *CSV* del programa *Dealmaster Pro* y de los ficheros de ayudas de texto (secciones 2.4 y 2.5).

3.3 Diseño de los componentes

A continuación se refleja un estudio de los diagramas de clases de cada uno de los componentes detallados en el apartado anterior. Con estos diagramas se representa la funcionalidad de cada uno de estos componentes. Dado el reducido tamaño del *software* a desarrollar, hay clases que engloban funcionalidad perteneciente a dos componentes distintos. Así, a lo largo de la explicación se detallará qué partes, principalmente métodos, son propios de cada uno de los componentes.

3.3.1 Diseño de la interfaz del componente Vista

En este apartado se detalla un boceto con la disposición de los elementos visuales principales de cada uno de los componentes que forman parte del componente Vista. Han sido diseñados con el objetivo de crear una interfaz agradable, sencilla, limpia y sobre todo, atractiva.

- **Pantalla Bienvenida**

  Esta pantalla presentará el título del software, los creadores y los agradecimientos como se puede observar en la Figura 20.

Figura 20. Pantalla Bienvenida

- **VistaMenú**

  Esta vista representa los diferentes casos de uso de los que dispone el usuario, además de la elección del nombre del jugador/a. El boceto de este componente se puede apreciar en la siguiente Figura.

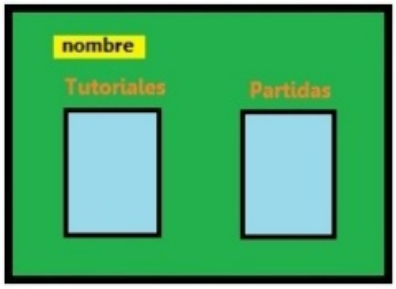


Figura 21. VistaMenú

- **VistaTutorial**

  En esta vista se lleva a cabo el caso de uso CU.01. El boceto de este componente se refleja en la siguiente Figura.

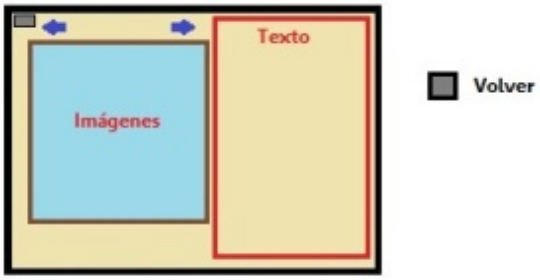


Figura 22. VistaTutorial

- **VistaPartida**

  En esta vista se llevan a cabo los casos de uso CU.02 y CU.03. Este componente tiene dos constructores diferentes, uno para cada caso de uso. Por este motivo se reflejan (Figura 23) los dos bocetos de este componente.

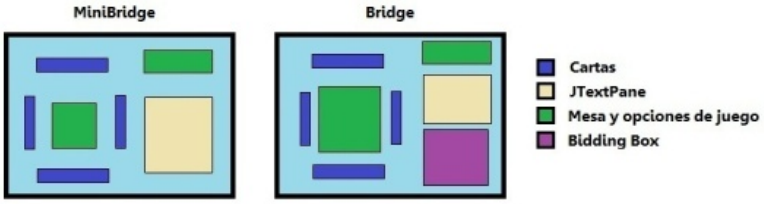


Figura 23. VistaPartida

3.3.2 Diagrama de clases del componente Vista

Este componente, como se ha visto en la sección 3.1, está formado por tres subcomponentes, *VistaMenu*, *VistaTutorial* y *VistaPartida*. Cada uno de estos tres subcomponentes será diseñado con una clase *Java*. La Figura 24 representa el diagrama de clases del componente Vista.

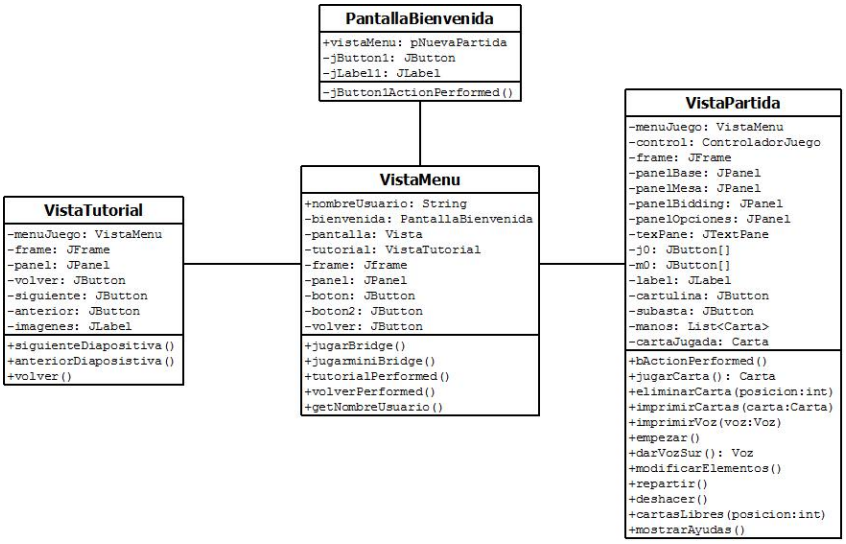


Figura 24. Diagrama de clases del componente Vista

A su vez habrá otra clase, *PantallaBienvenida*, que no tiene asignado el nombre de Vista por no ser un subcomponente de la misma. Se ha incluido en el diagrama para reflejar todas las clases creadas en este *software*.

La clase *VistaMenu* es la clase punto de inflexión de todas las vistas. Esta clase está unida bilateralmente a todas las demás, es decir, tiene como atributo todas las

demás clases que conforman el componente y a su vez estas últimas tienen como atributo la clase *VistaMenu*. De este modo se consigue que el usuario pueda volver al menú de juego para seleccionar otro caso de uso sin tener que volver a ejecutar el programa.

La clase *PantallaBienvenida* es la encargada de recibir al usuario y presentarle el nombre del proyecto y autores.

La clase *VistaTutorial* interactúa con el usuario reflejando y llevando a cabo el caso de uso CU.01. Por esta vista se reflejarán los distintos tutoriales de iniciación y aprendizaje del Bridge.

Por último está la clase *VistaPartida*, la más importante de todas las que forman este componente. Esta clase es la encargada de reflejar los elementos necesarios e interaccionar con el usuario para llevar a cabo partidas de miniBridge y de Bridge, es decir, los casos de uso CU.02 y CU.03 respectivamente. Esta vista se comunica con el subcomponente *ControladorJuego*. A continuación (sección 3.3.3) se representa el diagrama de clases de este subcomponente.

3.3.3 Diagrama de clases del subcomponente *VistaPartida*

Como se ha explicado anteriormente, esta clase se encarga de reflejar los eventos de la partida e interactuar con el usuario para llevar a cabo las manos. El siguiente diagrama explica las clases a las que está asociado este subcomponente y sus respectivos motivos.

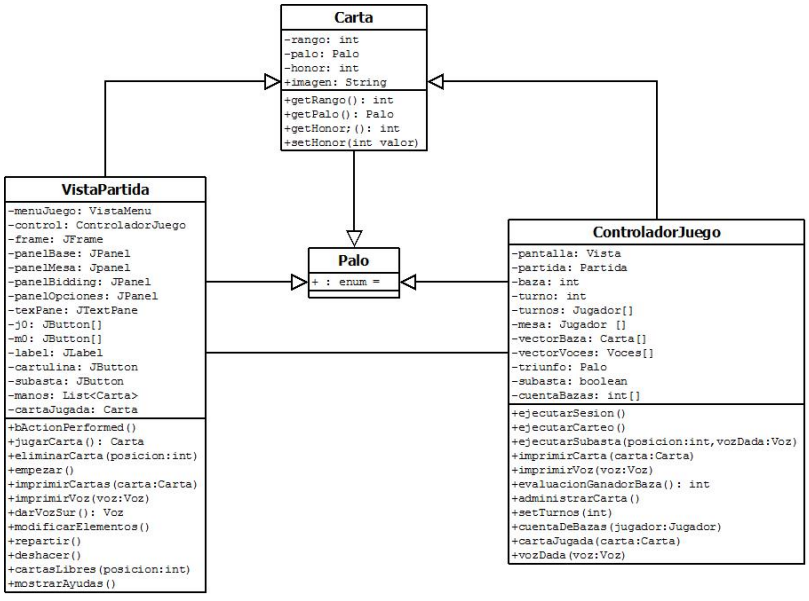


Figura 25. Diagrama de clase del subcomponente *VistaPartida*

En primer lugar se destaca que esta clase tiene dos constructores diferentes, uno para cada tipo de partida, miniBridge o Bridge. Los atributos principales de esta clase son de la librería *swing*, concretamente un *JFrame*, que hará de sustento de toda la vista, y un *JPanel* del mismo tamaño que el *JFrame* al que se incorporarán todos los de demás elementos. Esos elementos serán diferentes paneles secundarios, *botones JButtons*, vectores de *botones*, que simularán las cartas y voces, así como varios *JTextPanes* donde se reflejarán las ayudas y orientaciones escritas. Como atributos no visuales está la clase *Carta* en *listas* de la librería *LinkedList*. Con estas *listas*, el programa identificará qué carta está asociada a cada *JButton*, para coger así las imágenes de estas últimas y representarlas en sus respectivos *botones*.

Esta clase está unida bilateralmente a la clase *ControladorJuego*. De este modo pueden intercambiarse información y llamar ejecutar métodos el uno al otro de una manera más sencilla, por lo que cada uno de ellos tiene como atributo la otra clase.

Como métodos a destacar de la clase *VistaPartida* son *jugarCarta()*, *darVoz()*, *imprimirCarta()*, *imprimirVoz()* y *bActionPerformed()*. El método *jugarCarta()* es un método que interpreta qué *carta* se ha jugado en función del *botón* pulsado y su localización en la *lista*. Para ello cada *JButton* llevará incorporado un *Listener* para que el sistema identifique cuándo es ejecutado. Este procedimiento es exactamente el mismo que ocurre con el método *darVoz()*, pero en lugar de un vector de números enteros, es una matriz, identificando técnicamente qué *voz* ha dado el usuario. Una vez el *Listener* ha recibido la señal de implementarse, se llamará al método *bActionPerformed().* Cada *botón* tendrá su propio método *ActionPerformed()*, la "b" del principio es una letra esquemática de todos los *ActionPerformed*, que indica qué *botón* concreto ha causado el evento. El método *imprimirCarta()* viene con un parámetro de la clase *Carta* desde la propia clase *VistaPartida* o desde la clase *ControladorJuego*. Este método representa en la mesa de juego la *carta* que entra en el método, que es la que se acaba de jugar, bien por el usuario, bien por un jugador máquina. El mismo procedimiento se lleva a cabo con el método *imprimirVoz()*, que a excepción de ser una *carta*, es una *voz*.

3.3.4 Diagramas de clases del subcomponente Controlador Juego

El componente Controlador es el encargado de llevar a cabo correctamente toda la lógica de juego durante las partidas en función de los requisitos del *software* (sección 2.7). A continuación se detallan dos diagramas de clase de los subcomponentes del Controlador con la clase *VistaPartida*, facilitando así la comprensión de la lógica del programa llevada a cabo durante los casos de uso CU.02 y CU.03, ambos especificados en la sección 2.6 de este documento.

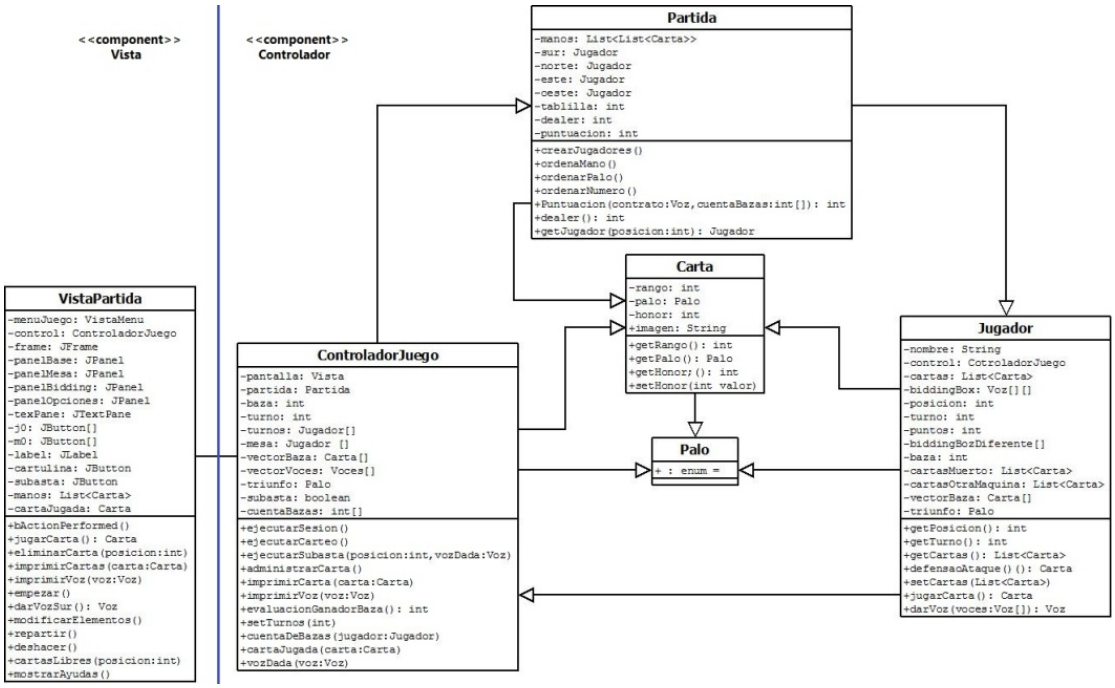


Figura 26. Diagrama de clases del subcomponente *ControladorJuego* 1.

El diagrama de clases anterior corresponde al subcomponente *ControladorJuego* con la clase *Carta*. El segundo diagrama es el mismo que el anterior, pero en lugar de representarse con la clase *Carta* se representa con la clase *Voz*.

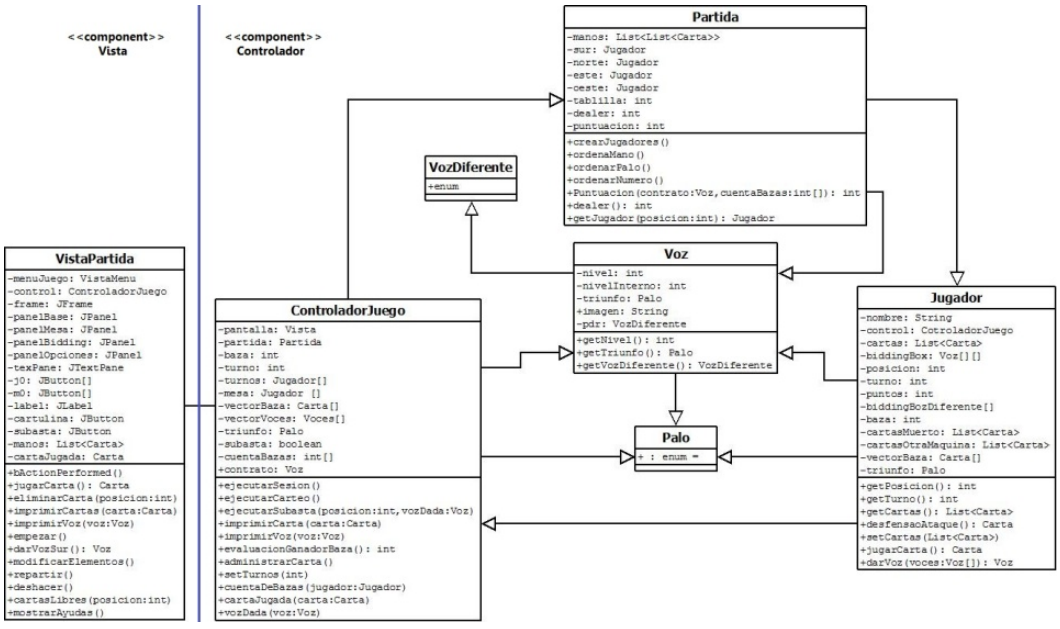


Figura 27. Diagrama de clases del subcomponente *ControladorJuego* 2.

Esta clase tiene dos constructores diferentes, uno para cada tipo de caso de uso. Como se ha mencionado anteriormente, la clase *ControladorJuego* está unida bilateralmente a la clase *VistaPartida*. A su vez la clase *ControladorJuego* tiene como atributo la clase *Partida* y esta última tiene como atributos cuatro objetos de la clase *Jugador*, Norte, Sur, Este y Oeste. Con las *cartas* y *voces* provenientes de los *jugadores*, *ControladorJuego*, implementa diferentes métodos de la clase *VistaPartida* para representar por pantalla los diferentes eventos. Concretamente con los métodos *imprimirCarta()* e *imprimirVoz()* (sección3.3.2), representando así las cartas y las voces dadas por los jugadores. De este modo se permite unir a los jugadores con el subcomponente *VistaPartida* de manera controlada.

Además de los anteriores atributos, la clase *ControladorJuego* tiene otros atributos principales para cumplir la función de llevar a cabo correctamente la lógica de las partidas. Como son los atributos *turno*, *baza*, *carta*, *voz, vectorBaza* (vector de *cartas* jugadas en una misma baza), *vectorVoces* (las voces que se irán dando) y el triunfo, que es de la clase *Palo*.

La clase *ControladorJuego* dispone de toda la lógica para llevar a cabo las partidas. A continuación se enumeran los métodos principales y más complicados:

- ***ejecutarSesion()***: Se encarga de empezar la mano y ejecutar la fase correcta de juego, el carteo o la subasta, en función del tipo de partida elegida previamente en *VistaMenu*.

- ***ejecutarCarteo()***: Este método es el encargado de llevar a cabo el carteo, ejecutándose trece veces por mano, una por cada baza. Para ello irá pidiendo cartas a los diferentes *jugadores* en función del turno. Cada vez que se hayan jugado las cuatro *cartas* de una baza llamará al método *evaluacionGanadorBaza()* .

- ***ejecutarSubasta()***: Este método es el encargado de llevar a cabo la subasta. A diferencia de método *ejecutarCarteo()*, éste sólo se ejecuta una vez, porque los turnos siempre son los mismos durante toda esta fase.

- ***evaluacionGanadorBaza()***: Se encarga de analizar el ganador de cada una de las trece bazas de las que se compone la mano, en función de las reglas de juego (Anexo III). Así se contabiliza el recuento de bazas de cada pareja. Una vez finalizado se llama al método *asignaciónTurnos()*.

- ***asignaciónTurnos()***: este método, como se ha explicado anteriormente, es llamado cuando el método de *evaluaciónGanadorBaza()* ha identificado el *jugador* ganador. Este método es el encargado de asignar los turnos a los *jugadores* para la siguiente baza en función del ganador de la misma. El ganador de la baza tendrá el primer turno en la siguiente.

3.3.5 Diagrama de clases de los subcomponentes JuegoBridge y Partida

La clase *JuegoBridge*, como se ha detallado anteriormente (sección 3.2), es un subcomponente del componente Controlador. Esta clase es la encargada de crear todos los objetos necesarios para llevar a cabo el cualquier tipo de partida, es decir, las *cartas*, la *baraja*, las *voces*, el *bidding box*, las *manos*, y la *partida*. Las *manos* serán generadas a partir de la información del componente Modelo (sección 3.3.6). Como se ha explicado anteriormente, dado el reducido tamaño del *software,* el componente Modelo estará integrado dentro de la clase *JuegoBridge.*

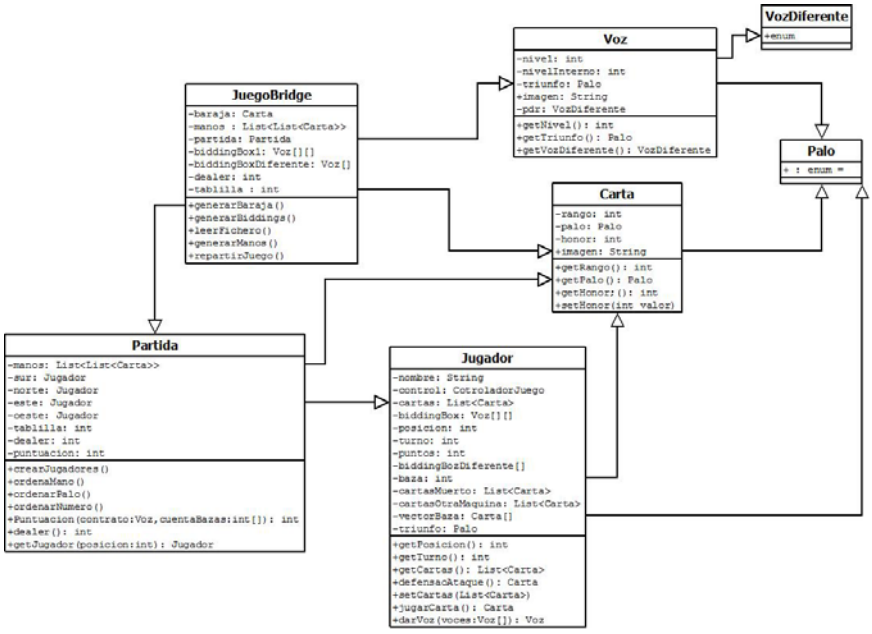


Figura 28. Diagrama de clases del componente Controlador

En la Figura 28 se pueden observar los subcomponentes de la clase controlador. No se ha incluido la clase *ControladorJuego*, porque ya se incluyó anteriormente.

Para llevar a cabo la creación de estos objetos en *JuegoBridge*, se diseñaron los siguientes métodos:

- ***generarBaraja()***: este método se encarga de crear las *cartas* y la *baraja*.
- ***generarBiddings()***: este método se encarga de crear las *voces* y el *bidding box*.
- ***generarManos()***: en función de los datos del Modelo, se distribuyen las *cartas* en *manos*.
- ***repartirJuego()***: se encarga de crear el objeto *partida* con la clase *Partida*. Para ello lleva cómo parámetros las manos.

Una vez la clase *JuegoBridge* ha generado los objetos necesarios para cualquier mano se crea el objeto *partida* con la clase *Partida*. Esta clase *Partida* crea los *jugadores* en función de los objetos creados en *JuegoBridge*. Para ello genera los cuatro *jugadores* y les asigna la *mano* correcta, previamente creada en la clase *JuegoBridge*. A la clase *Partida* llegan como parámetros las *manos* y tiene incorporados los siguientes métodos:

- ***crearJugadores()***: este método es el encargado de crear los cuatro *jugadores*, Norte, Sur, Este y Oeste.
- ***asignarManos()***: este método se encarga de asignar las *manos*, provenientes de *JuegoBridge*, a sus respectivos *jugadores*.
- ***ordenarManos()***: este método ordena las *manos*.

- **Clase *Carta***

La clase *Carta* es la encargada de generar los 52 objetos *carta* de la *baraja*, es elemento primordial de cualquier tipo de partida. Esta clase tiene como atributos el *rango* de la clase mediante un *(int),* el *palo* de la clase *Palo*, el honor de la clase *(int)* y las imágenes como cadena de *String*. Los métodos que incorpora son *get's*, de todos los atributos, para poder comparar unas cartas con otras, y un método *setHonor()* que incluye como parámetro un número entero (*int)* para modificar el atributo *honor* de cada *carta*. Este último atributo, el *honor*, sirve para que un *jugador* máquina calcule de una manera sencilla, cuantas cartas se hayan todavía en juego por encima de esa carta. Por ello, una vez se van jugando las cartas, el *ControladorJuego* modificará el *honor* de cada carta todavía en juego. El atributo *rango*, es el valor inmutable, con el cual se comparan realmente unas cartas con otras de la misma baza, en el método *evaluarGanadorBaza()* de la clase *ControladorJuego* (sección 3.3.4) .

- **Clase *Voz***

La clase *Voz* es la encargada de generar los 38 objetos *voz* del *bidding box*, elemento indispensable para llevar a cabo la subasta en una partida de Bridge. Esta clase tiene como atributos el *nivel (int)*, *nivelInterno (int)*, el *triunfo* de la clase *Palo*, el atributo *pdr* de la clase *VozDiferente* y las imágenes como cadena de *String*. Los métodos que incluye son *get's* para obtener los atributos de cada *voz* para poder ser evaluadas y comparadas con otras *voces* previamente dadas.

Cabe destacar que las clases *Carta* y *Voz* poseen un constructor nulo para permitir crear *cartas* y *voces* auxiliares en las demás clases, facilitando así los procesos de diferentes métodos.

- **Clase *Palo***

La clase *Palo*, es una clase pública, la cual tiene como único atributo una enumeración de los cuatro palos de la baraja y el palo Sin Triunfo (sección 1.4). Dicha enumeración es la siguiente; *PICAS, CORAZONES, DIAMANTES, TREBOLES, ST.*

- **Clase *VozDiferente***

La clase *VozDiferente* es una clase pública que, al igual que la clase *Palo*, tiene como único atributo una enumeración de las diferentes voces especiales. La enumeración es la siguiente; *PASO, DOBLO, REDOBLO, AUXILIAR*. Las voces de *doblo* y *redoblo* son incluidas para posibles mejoras del programa.

- **Clase *Jugador***

Habrá cuatro objetos diferentes de la clase *Jugador*, Norte, Sur, Este y Oeste. Cada uno de estos jugadores recibirá trece *cartas* de la baraja. Esta clase es la encargada de llevar a cabo la lógica de selección de los objetos *carta* y *voz*, explicados anteriormente, en función del análisis de juego (sección 2.10). Para ello cada *jugador* controlado por el propio programa, evaluará el atributo *honor* de la *carta* y no el *rango*. Esto se argumenta en que por ejemplo un cinco de tréboles, al final de la partida puede ser la carta de trébol más alta todavía en juego, por ello, para el jugador será como un *As* a todos los efectos. Si por ejemplo, es la segunda mayor de las que quedan será como un rey también a todos los efectos. Se aclara que el rango de la carta, es utilizado por el Controlador de juego para evaluar el ganador de la baza, de una manera fiable y segura ya que el rango es un atributo inmutable a lo largo de la partida. Gracias al atributo *honor* de la carta se facilitará el proceso de implementación.

3.3.6 Componente Modelo

Como se ha detallado en el diagrama de componentes, apartado 3.2 de este capítulo, el componente Modelo está formado por el subcomponente Gestor Partidas. Este subcomponente está formado por dos métodos que se encargan de leer la información de los ficheros de *CSV* de *DealMasterPRo* y los ficheros de texto, ambos especificados en las secciones 2.4 y 2.5 respectivamente. Ambos métodos están integrados en el subcomponente *JuegoBridge* debido a la pequeña magnitud del *software*.

El método *leerFichero()* es el encargado de leer el fichero *CSV* y la función del método *leerAyudas()* es leer las ayudas que posteriormente serán representadas en el *JTextPane* del subcomponente *VistaPartida* (sección 3.3.3).

Estos métodos anteriormente mencionados, llevarán a cabo sus respectivas funciones gracias a dos clases obtenidas del *API* (26) de *Java*, cuyos nombres son *FileReader* y *BufferedReader.* Una vez leídos los ficheros se interpretará la información según los parámetros definidos en el análisis de ambos tipos de ficheros.

3.3.7 Diseño de los iconos las clases Carta y Voz.

En este apartado se detalla el diseño de las imágenes de los 52 objetos carta y los 38 objetos *voz* que se implementarán sobre sus respectivos *JButton* de la clase *VistaPartida*.

- ***Carta***

  Las imágenes de la cara se diseñaron según los requisitos RN.02, RN.03, RN.04 y RN.05, especificados en la sección 2.7.3, para que la estética de la interfaz fuese moderna (sección 2.2). Este diseño se puede observar en la Figura 29. El tamaño de estas imágenes es de 70x96 píxeles y su orientación será la misma para todos *botones*, es decir, no habrá cartas en horizontal.

  Las imágenes del dorso de las cartas se diseñaron en azul y matices blancos como se puede apreciar en la Figura 29. Su tamaño es más reducido que la cara de las mismas, para no saturar la interfaz. Este tamaño de los dorsos es de 36x60 píxeles.

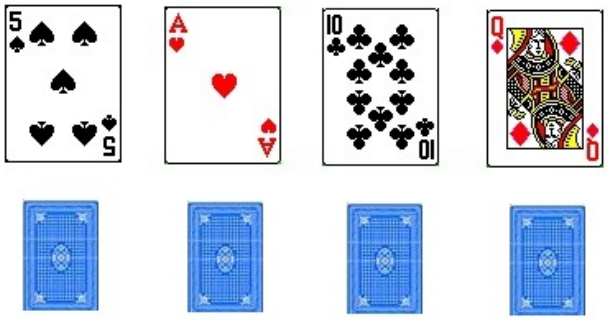


Figura 29. Diseño de las cartas

- ***Voz***

  Las *voces* tienen dos formatos diferentes, uno para la mesa, y otro para el *bidding box*, ambos son los requisitos RN.10 y RN.11 (sección 2.7.3).

  Las voces de nivel del *bidding box* tendrán un número indicativo del nivel y del palo al que hacen referencia, Figura 30. El tamaño de las voces será del tamaño real de un *bidding box*, es decir, de 40x27 píxeles. La *voz*

especial de *PASO*, será implementada con un tamaño mayor ya que es la más importante y la más solicitada. Su tamaño será de 70x40, como se observa en la 30. Las otras *voces* especiales, *Doblo y Redoblo*, serán diseñadas y representadas, pero estas carecen de funcionalidad (RF48).

Figura 30. Diseño del *bidding box*

Las *voces* reales que se representarán sobre la mesa de juego fueron diseñadas en blanco pero con el nivel y palo que representan o en su defecto solo el *PASO*. Estas imágenes tienen tamaños diferentes en función del nivel de la *voz*. Para las *voces* comprendidas entre los niveles 1 y 3, ambos incluidos, el tamaño de las voces es de 50x80 píxeles. Para los niveles 4 y 5 de 50x70 píxeles. Por último, para los niveles 6 y 7 de un tamaño de 50x60 píxeles. La *voz* especial de *PASO* tiene un tamaño de 50x50 píxeles. Estas imágenes se pueden observar en la Figura 31.

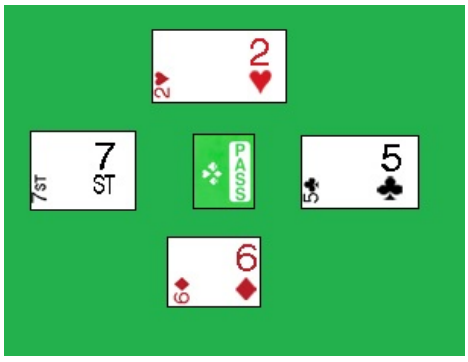


Figura 31. Diseño de las voces del *bidding box*

3.4 Diagramas de Secuencia

A continuación se representan los diagramas de secuencia correspondientes a los casos de uso especificados y analizados en la sección 2.6. En estos diagramas no se incluyen todos los mensajes que se intercambian entre objetos y clases, sino únicamente los más relevantes para facilitar la comprensión global de dicho diagrama. Para el desarrollo de los mismos se ha utilizado el programa *Microsoft Visio 2010* (27).

3.4.1 Diagrama del caso de uso CU.01: Mostrar tutoriales de Bridge

Este caso de uso muestra los tutoriales de miniBridge y Bridge con el objetivo de familiarizar al usuario con el juego y poder entender mejor las partidas y sus fases.

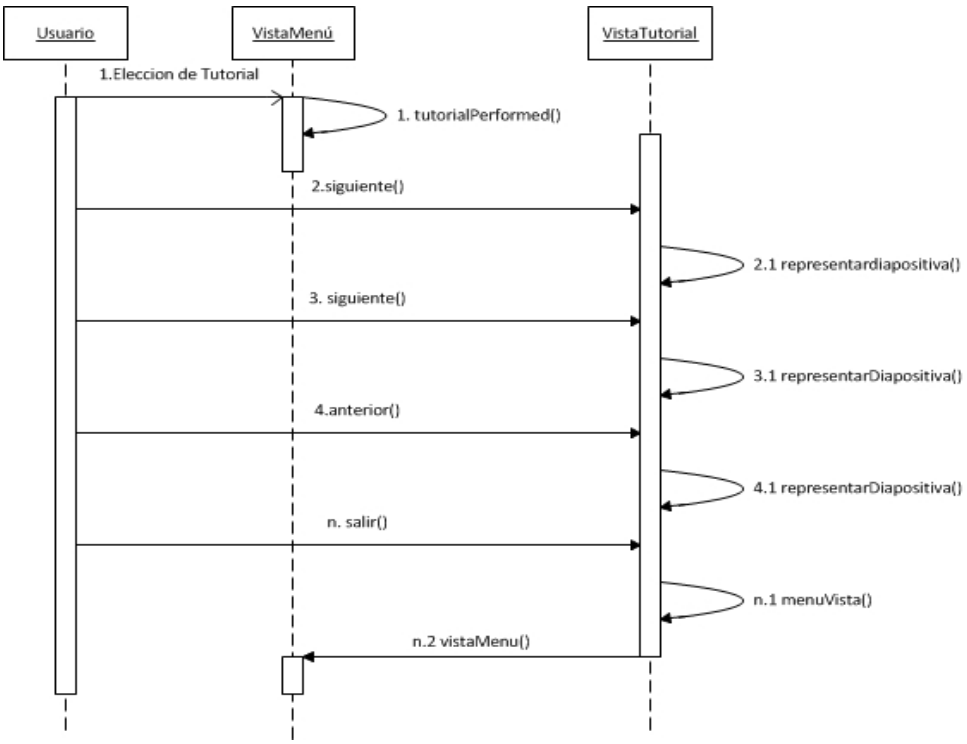


Figura 32. Diagrama de secuencia CU.01

En primer lugar el usuario elige en subcomponente *VistaMenu* la opción de "Tutorial de Bridge o miniBridge" llamando así al método *tutorialPerformed(). Este* método llama crear a *VistaTutorial* y cierra *VistaMenu*. Una vez dentro de la *VistaTurorial*, el usuario va cambiando de imagen para familiarizarse con el juego. Indistintamente pude ir hacia adelante con el método *siguiente()*, como hacia atrás, con el método *anterior()*, representando de nuevo la imagen anterior. Una vez el usuario considere oportuno abandonar los tutoriales, con el método *salir()* volverá a la *VistaMenu*.

3.4.2 Diagrama del caso de uso CU.02: Jugar partida MiniBridge.

Este caso de uso interactúa con el usuario para llevar a cabo partidas de miniBridge. Este caso de uso se ha dividido en tres partes para mejorar la explicación del mismo. Estas tres partes son el reparto inicial, el carteo y por último, posteriores repartos y la salida.

1º Reparto inicial.

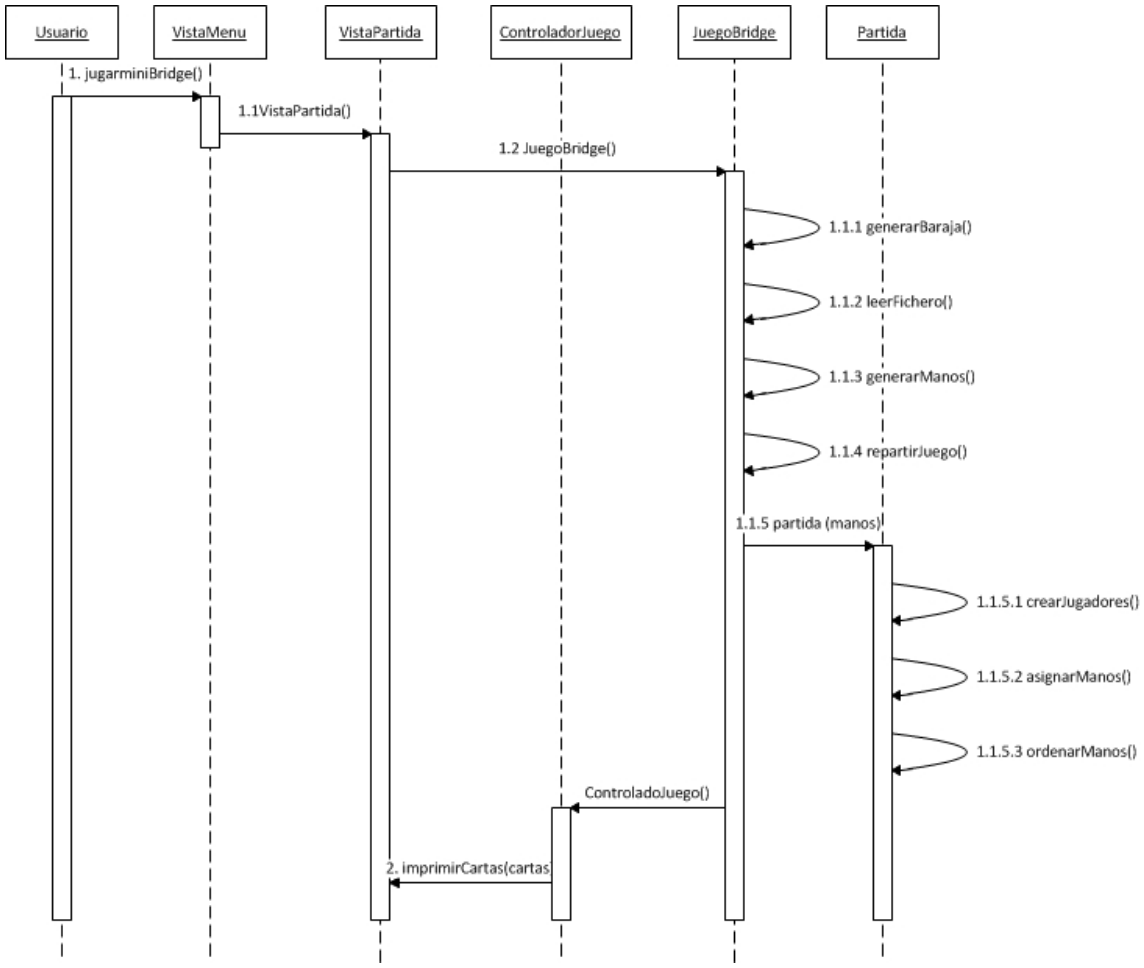


Figura 33. Diagrama de secuencia CU.02 Reparto

En primer lugar, el usuario selecciona en *VistaMenu* la opción de jugar una partida de miniBridge. Esta acción llama al método *jugarminiBridge()*. Este método crea el objeto *VistaPartida* con el primero de sus dos constructores (sección 3.3.2) y cerrando *VistaMenu*.

Una vez creado el objeto *VistaPartida,* también se crea un objeto de la clase *JuegoBridge*. Esta última clase, como se especifica en la sección 3.3.5, crea todos los materiales necesarios, gracias a los métodos, *generarBaraja()*, *leerFichero()*, *generarManos()* y *repartirJuego()*. A su vez, *JuegoBridge,* crea la *partida* (sección 3.3.5). La clase *Partida*, se encarga de crear los cuatro jugadores y asignar y ordenar las manos. Una vez llegados a este punto, se crea un objeto *ControladorJuego* que llama al método *imprimirCartas()* de *VistaPartida* para mostrar por la interfaz las cartas de Norte y Sur.

2º Carteo

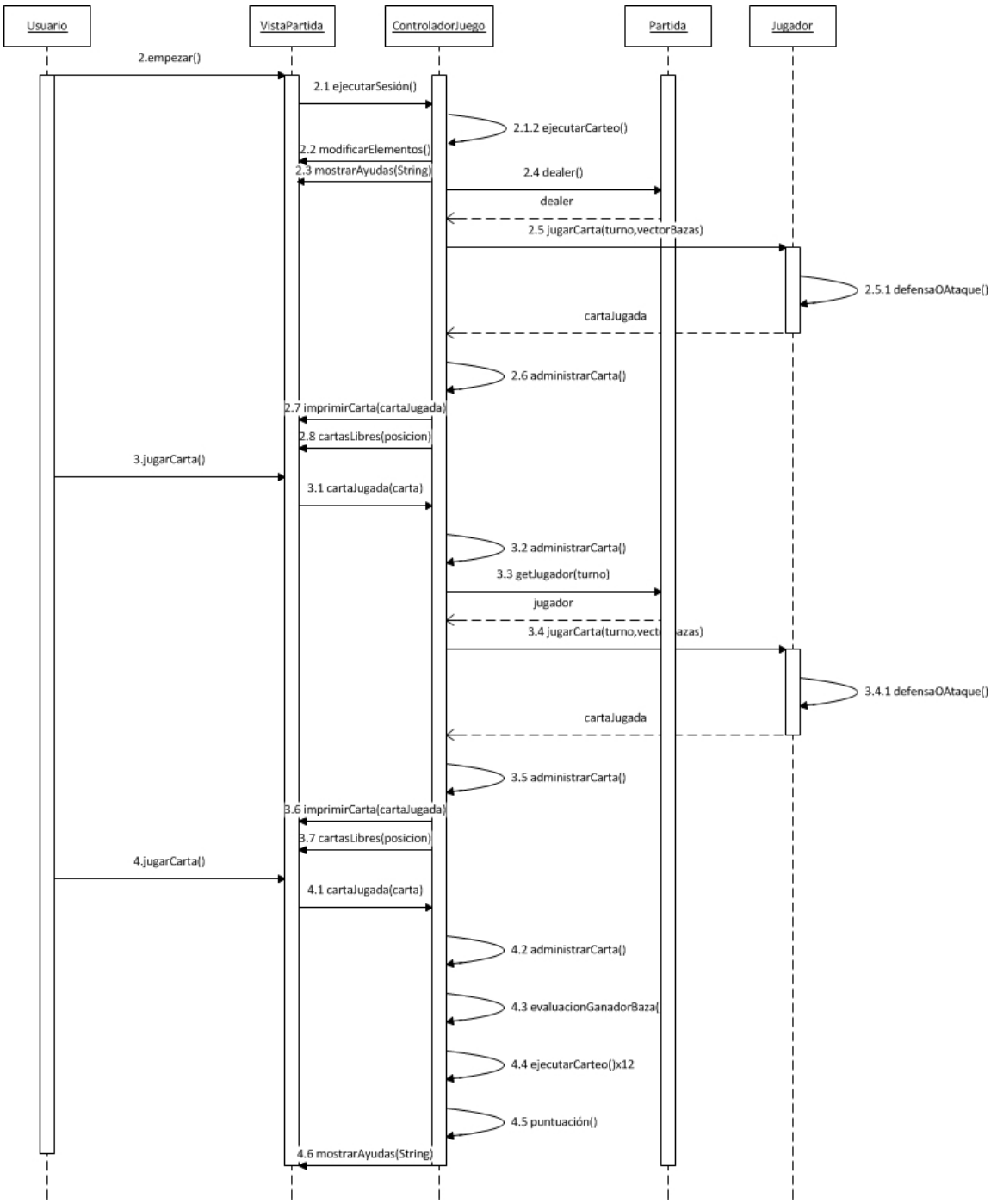


Figura 34. Diagrama de secuencia CU.02 Carteo

Una vez está todo preparado para empezar la mano, el usuario ejecuta el método *empezar()*, y este último implementará el método *ejecutarSesión()* de *ControladorJuego*. Este último método identifica qué fase tiene lugar a continuación. En este caso la siguiente fase es el carteo, por lo que ejecutará el método

ejecutarCarteo(). Posteriormente se muestran por pantalla las ayudas para la mano sobre el *JTextPane* de *VistaPartida,* gracias al método *mostrarAyudas()*. A partir de aquí, *ControladorJuego* ira administrando la información tanto de *partida* como de *VistaPartida* y asignado los turnos de juego.

Al terminar de mostrar las ayudas*, ControladorJuego*, llama al método *getJugador() de Partida* para coger el *jugador* Oeste. Una vez llamado el *jugador* con el primer turno de carteo, *ControladorJuego* le pide que juegue una carta con el método *jugarCarta().* Para que el *jugador* pueda tomar la decisión de llevar a cabo una jugada coherente, la llamada a este método tiene dos parámetros, el *turno* en el que se encuentra el *jugador* dentro de la baza mediante un entero *(int)* y las *cartas* que ya han sido jugadas en la baza mediante un vector de cartas. Si el *turno* de juego dentro de la baza es el primero, es decir, es el primero que juega carta en esa baza, el *jugador* implementará el método de *ataque()*, si por el contrario, no dispone del primer *turno* de juego, éste implementara el método *defensa()*. Una vez el *jugador* ha elegido la *carta* oportuna, *ControladorJuego* administra esa *carta*, modifica el turno de juego y la representa en *VistaPartida*, gracias al método *imprimirCarta()* que tiene como parámetro la *carta* jugada y el *turno*. Si el último *jugador* que jugó carta fue un *jugador* máquina, le toca el turno a uno de los *jugadores* que maneja el usuario, el Muerto o el declarante. En la primera baza el Muerto es el *jugador* que posee el segundo turno.

El siguiente turno lo posee uno de los *jugadores* que maneja el usuario, por lo que *ControladorJuego* implementa el método *cartasLibres()* de *VistaPartida*. Este método lleva como parámetro la posición *(int)* del *jugador* y *VistaPartida* se encarga de liberar los *botones* pertinentes de ese *jugador* para que las *cartas* puedan ser jugadas reglamentariamente. De este modo *ControladorJuego* se cerciora de que se respeten estrictamente los turnos de juego dentro de cada baza y de que se eviten renuncios.

Este procedimiento de elección de cartas se repite cuatro veces por turno, dos jugadores máquina y dos jugadores usuario. Una vez terminada la baza, *ControladorJuego* analiza el *jugador* ganador en función del *vectorBaza* y reasigna los turnos para la siguiente. Este proceso completo se repetirá doce veces más durante la partida ya que hay trece bazas.

Una vez finalizada la mano, el programa representa por pantalla la conclusión final. Esta conclusión es en función del número bazas logrado por el usuario respecto del número exigido inicialmente. En función de estos dos factores anteriores se calcula la puntuación exacta de Bridge (sección 2.10).

3º Siguientes repartos y Salida

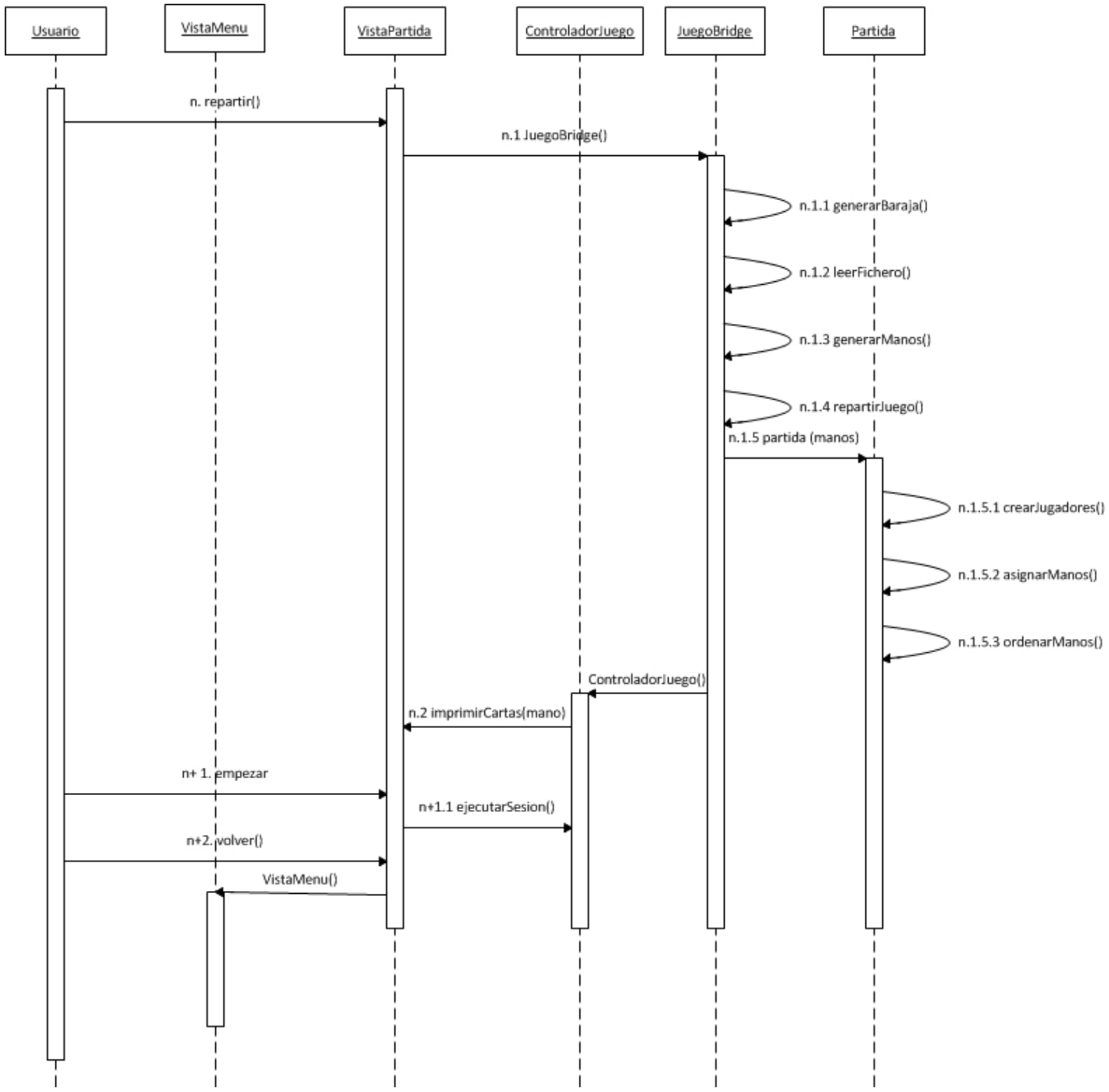


Figura 35. Diagrama de secuencia CU.02 Siguientes repartos y salida

Una vez ha concluido la mano, el usuario tiene la opción de jugar otra mano nueva, repetirla o volver a *VistaMenu* para seleccionar otro caso de uso.

Suponiendo que el usuario escoge jugar otra mano, el procedimiento llevado a cabo por el *software* es el mismo que en el apartado 1º de este diagrama, el reparto. Se recalca que todo el componente Controlador, es decir, los subcomponentes y clases *ControladorJuego*, *JuegoBridge* y *Partida* se eliminan y se crea un componente entero nuevo. Se ha diseñado de este modo para asegurarse de que todos los objetos tienen los valores iniciales para proceder correctamente a una nueva mano.

3.4.3 Diagrama del caso de uso CU.03: Jugar partida Bridge.

Este caso de uso interactúa con el usuario para llevar a cabo partidas de Bridge. De este modo el usuario se familiariza con las dos fases del juego, subasta y carteo. Este caso de uso se ha dividido en cuatro partes para mejorar así la explicación del mismo. Estas cuatro partes son el reparto inicial, la subasta, el carteo y por último, posteriores repartos y la salida.

1º Reparto

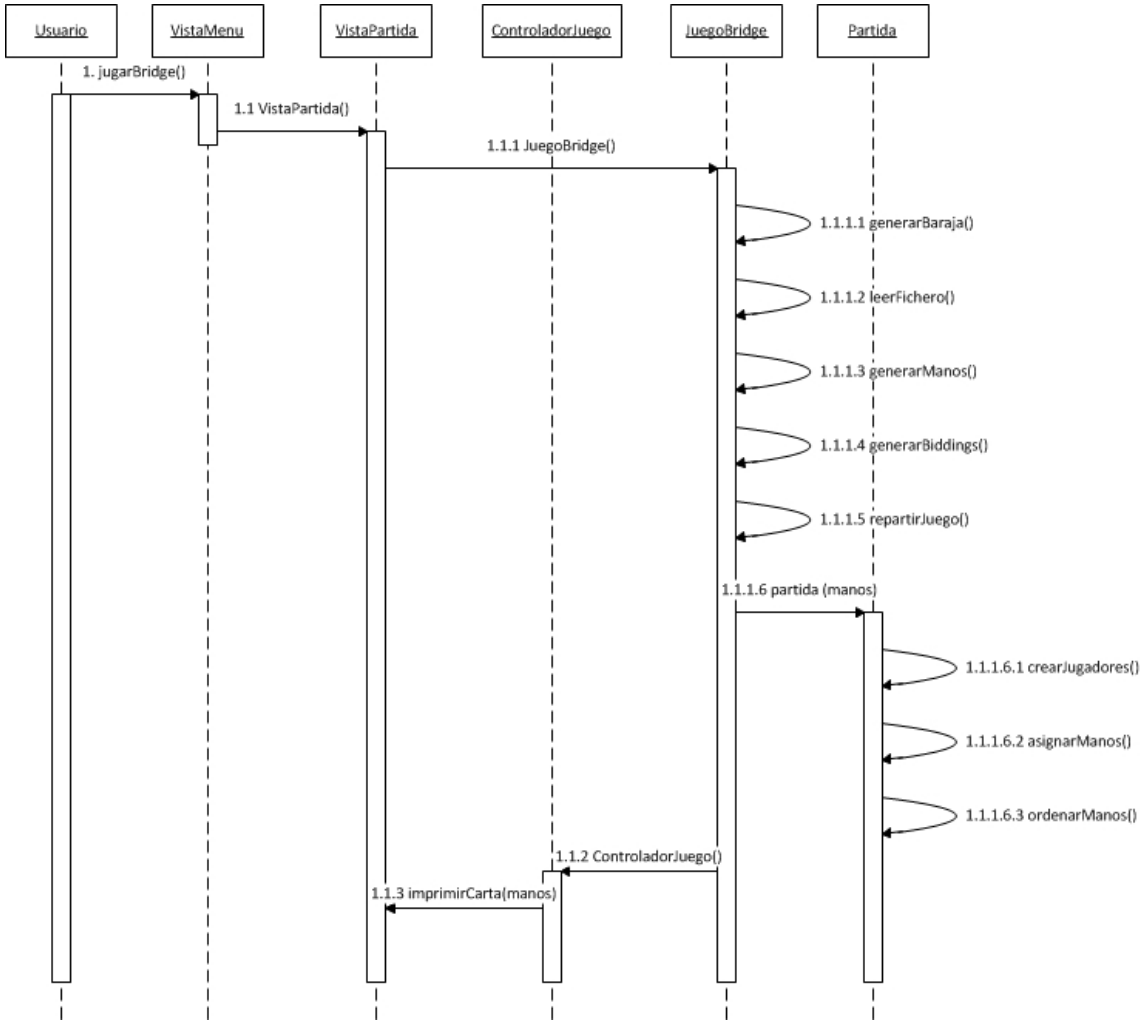


Figura 36. Diagrama de secuencia CU.03 Reparto

El procedimiento de reparto de este caso de uso es el mismo que el llevado a cabo en caso de uso CU.02, del apartado anterior, a excepción de que ahora se crea el *bidding box* para la fase de subasta.

2º Subasta

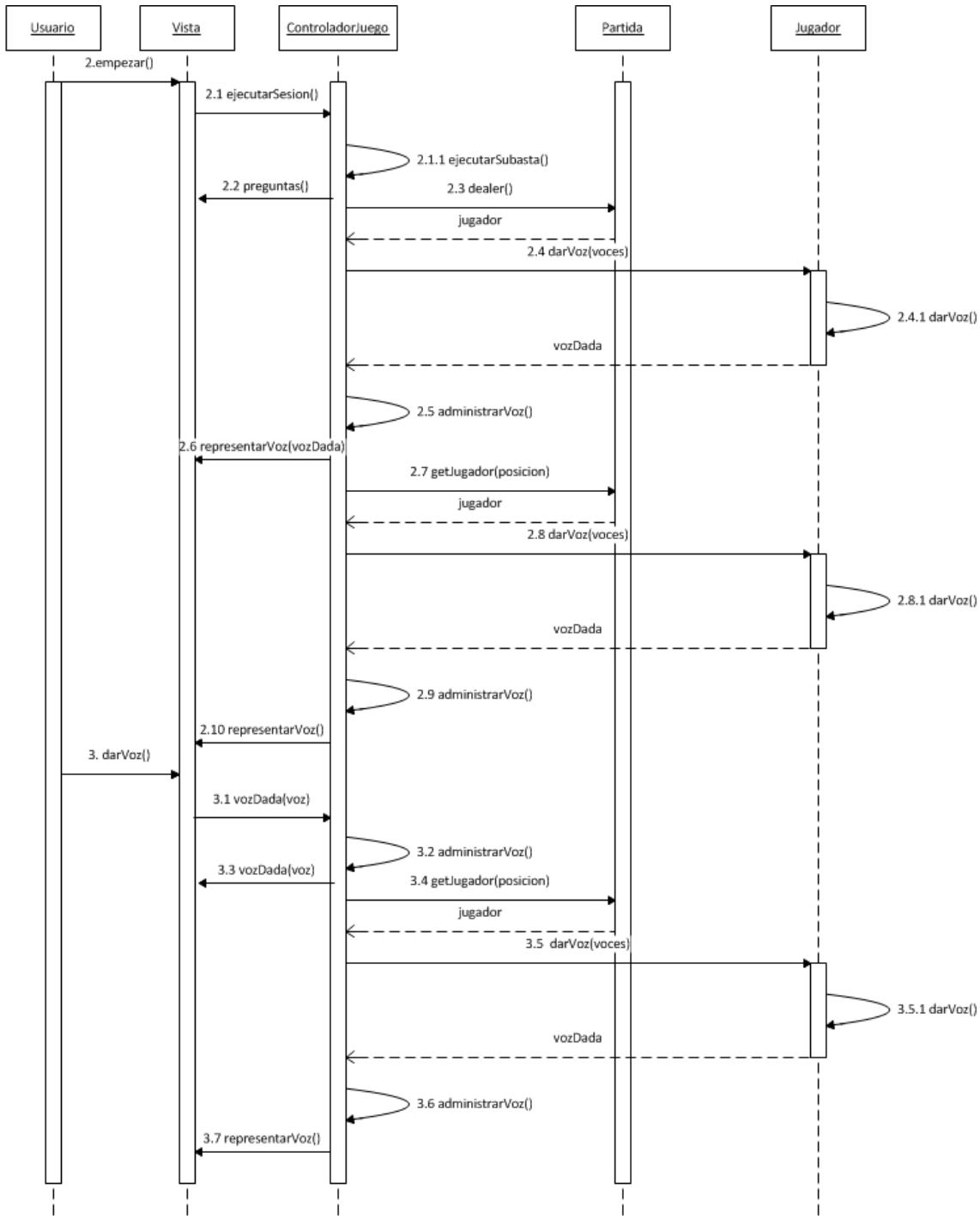


Figura 37. Diagrama de secuencia CU.03 Subasta

La fase de subasta empieza cuando el usuario considere oportuno, ejecutando el método *empezar()* de *VistaPartida*. Este método implementa el método *ejecutarSesion()* de la clase *ControladorJuego*. Este último procederá a llevar a cabo la subasta llamando al método *ejecutarSubasta()*. Una vez empieza la fase de subasta, si

las ayudas están activadas (por defecto), el *software* efectuará dos preguntas técnicas de ayuda al usuario para ver si es capaz de llevar a cabo la subasta. Estas dos preguntas son las más elementales de la subasta en el Bridge y son; *¿Cuántos puntos de honor tienes en tu mano?* y *¿Cómo es tu mano?* Ambas preguntas serán respondidas correctamente si el usuario ha leído los tutoriales de Bridge. Mientras no conteste a estas dos preguntas básicas, el programa no permitirá que el usuario pueda proceder a la subasta. Estas preguntas serán expuestas por pantalla con diversos *JLabels*, *JButtons* y un *JTextPane*, e implementados gracias al método *preguntas()* de *VistaPartida*. Una vez ha respondido correctamente, *ControladorJuego* llama al *jugad*or de *Partida* que es el *dealer* de la subasta. Para este diagrama concreto se ha supuesto que el jugador *dealer* es el *jugad*or Norte. Una vez llamado le solicita que elija una *voz* con el método *darVoz()*. Para los *jugadores* den una voz apropiada, el método tiene como parámetro un vector de *voces*. Dentro del *jugador* se implementa el método *elecciónVoz()* para analizar qué jugador ha dado cada voz del vector y por qué motivo, pudiendo así elegir la *voz* más apropiada. Una vez elegida la *voz*, *ControladorJuego* la administra y la imprime por pantalla con el método *imprimirVoz()*, con dicha *voz* como parámetro. A continuación *ControladorJuego* analiza el siguiente *jugador* en turno y vuelve a repetir el procedimiento de llamar a ese nuevo *jugador* por medio de *Partida*.

Una vez han dado voz los jugadores anteriores, tiene el turno el usuario. Este llamará a un método al método *darVoz()* de *VistaPartida*, mediante el pulsado con el ratón sobre uno de los *botones* que representan el *bidding box*. *ControladorJuego* se encarga de analizar la *voz* dada y representarla por pantalla mediante los *analizarVoz()* e *imprimirVoz()* respectivamente. Los siguientes procedimientos son los mismos que los explicados a en el párrafo anterior.

Una vez ha habido tres *voces* de *PASO* seguidas, la subasta ha finalizado. Cuando finaliza la subasta se estudia si ha sido correcta en función de qué jugador sea el declarante. La subasta no será correcta cuando la pareja declarante son los jugadores máquina. De no ser correcta el programa manda repetir la subasta volviendo al punto inicial, eliminando todas las voces dadas y restableciendo los turnos y los vectores. Si la subasta es correcta, *ControladorJuego* analiza la última *voz* de nivel dada para designarla como el contrato definitivo.

Una vez analizado el contrato, *ControladorJuego* modificará sus atributos de *triunfo* de la clase *Palo*, y *númeroBazas (int)* prometidas por la pareja declarante.

La siguiente fase es el carteo, cuyo procedimiento es exacto al de miniBridge con la diferencia de que las bazas y triunfo han sido elegidas en la subasta.

3º Carteo

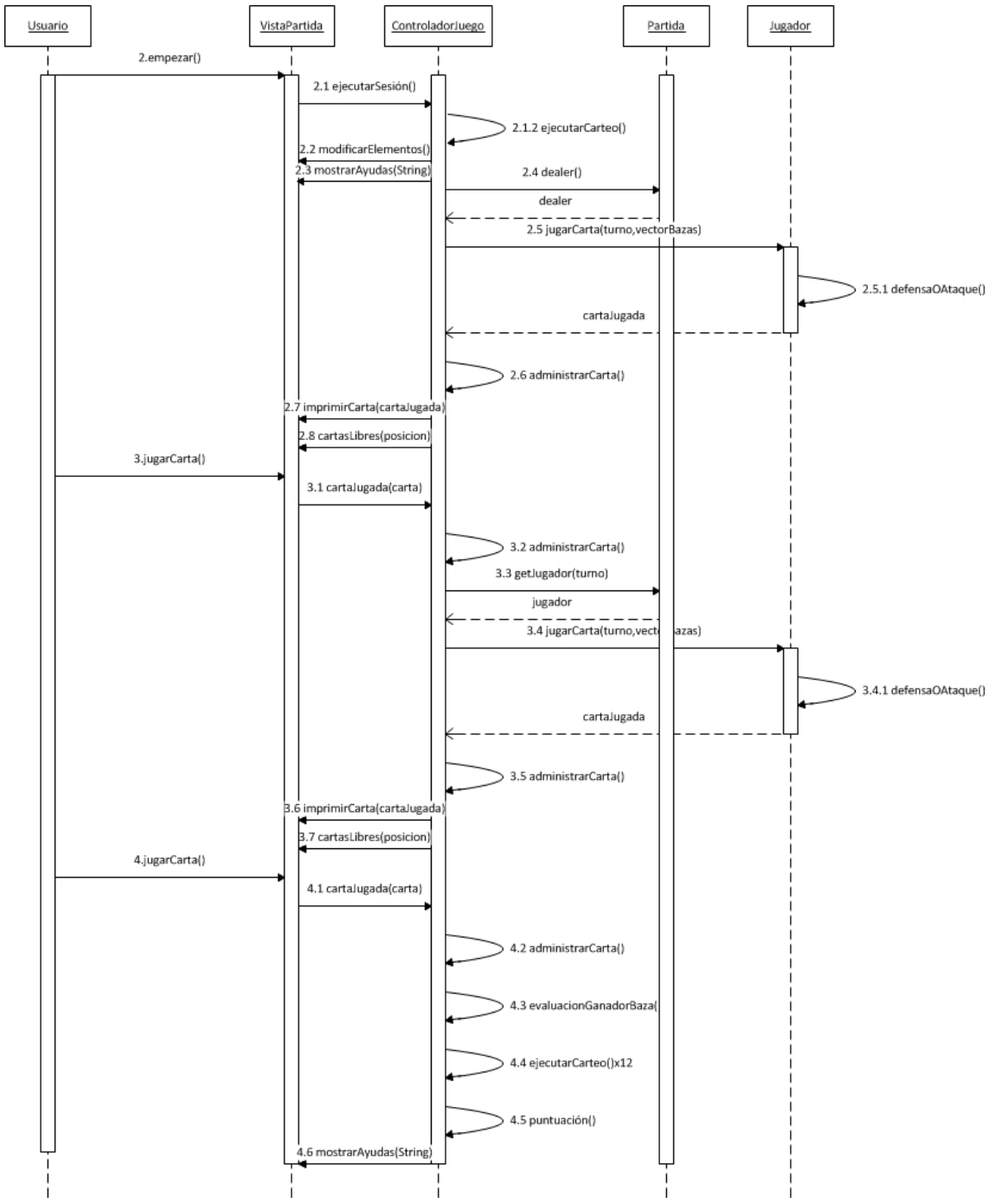


Figura 38. Diagrama de secuencia CU.03 Carteo

Esta fase de este caso de uso, como se ha explicado anteriormente, es un *<<include>>* del caso de uso CU.02. Por lo que a partir de aquí sucede exactamente lo mismo que en el diagrama de secuencia del caso anterior, concretamente durante el carteo.

4º Siguientes repartos y Salida

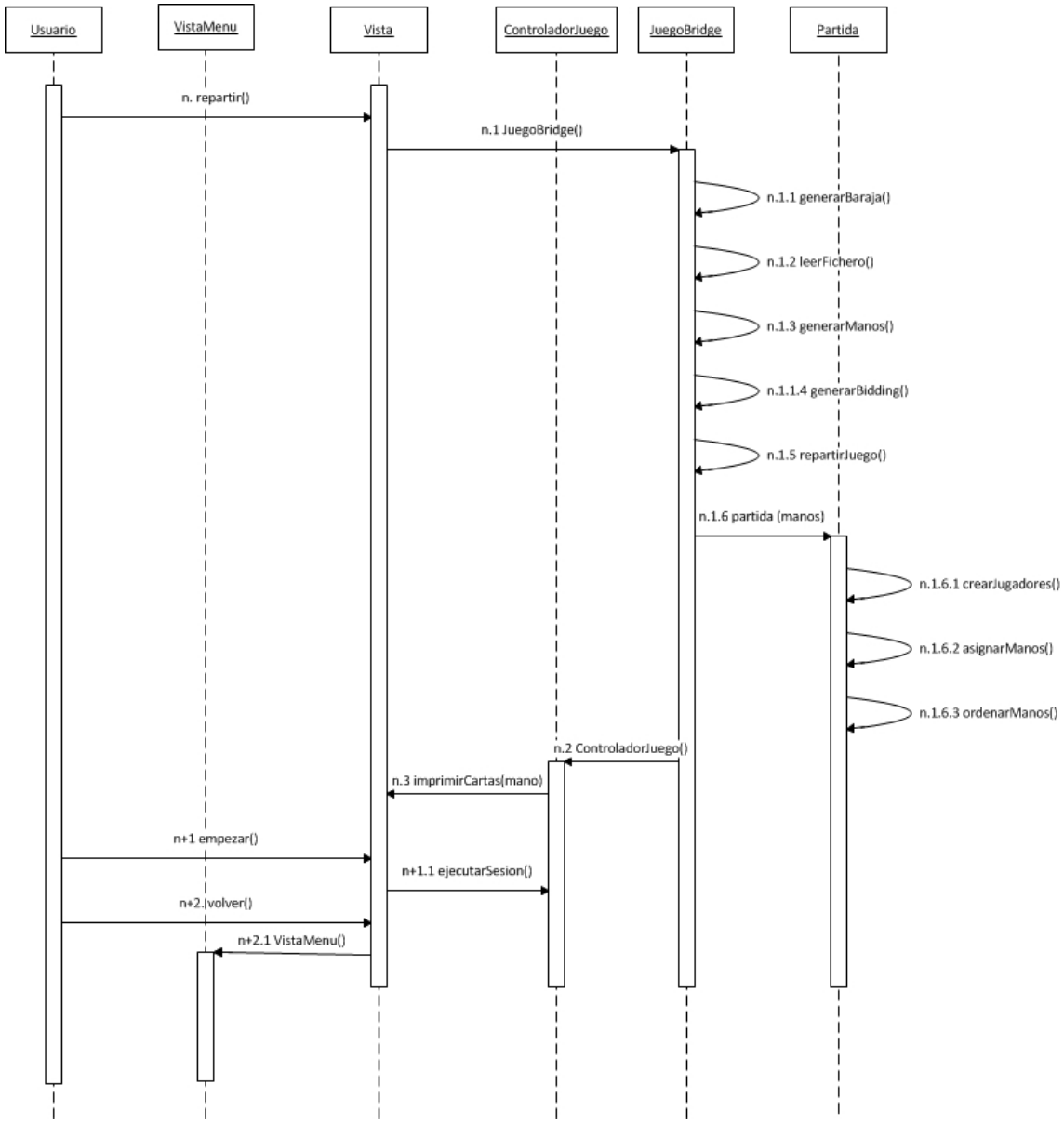


Figura 39. Diagrama de secuencia CU.03 Siguientes repartos y salida

Esta fase es la misma que la del caso de uso CU.02 , detallada en el apartado anterior, exceptuando como se ha mencionado anteriormente, que el subcomponente *JuegoBridge* crea la el *bidding box*. También se destruye y crea de nuevo todo el componente Controlador.

4. IMPLEMENTACIÓN DEL SOFTWARE

En este capítulo se recogen todos los detalles relacionados con el proceso de desarrollo de implementación del *software* en base al diseño, capítulo 3. Por ello se detallan los procesos de codificación de los subcomponentes (sección 4.1, 4.2 y 4.3). Posteriormente se incluyen los resultados de las pruebas de aceptación definidas en el plan de pruebas (sección 2.7).

4.1 Proceso de codificación del componente Vista

En este apartado se especifican los detalles más importantes del proceso de codificación de cada uno de los componentes del *software* y los principales problemas surgidos durante el mismo.

Cabe destacar que fue un acierto utilizar Java, que gracias a la librería *javax.swing*, obtenemos todos los *JFrames, JPanels, JButtons, JLabels, JTextFields* y *JTextPanes*.

4.1.1 Subcomponente Vista Partida

Para la creación de *VistaPartida* se optó por no utilizar la herramienta *Palette* para construir interfaces que facilita *NetBeans*. Esta herramienta no nos permitía colocar un elemento en un sitio estático en la interfaz mientras tenían lugar diferentes eventos. Por ello se implementaron uno a uno cada uno de los elementos visuales de las interfaces, definiendo su forma, tamaño y coordenadas estáticas.

- **Constructor**

  Esta clase tiene dos constructores diferentes, una para cada caso de uso, CU.02 y CU.03. En función del constructor implementado, se crearán diferentes elementos.

- **JFrame y JPanel**

  Cada Vista tiene implementada como atributo un *JFrame* que hace de soporte general para que puedan aparecer los elementos. Una vez creado el soporte general se incorporó encima un *JPanel*, del mismo tamaño que el *JFrame*, Figura 40, para poder incorporar todos los demás elementos.

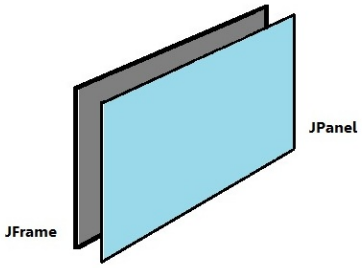


Figura 40. Disposicion de JFrame y JPanel

Para poder elegir la disposición que ocuparán todos los objetos de la Vista se implementa nulo el *Layout* de los objetos *JPanel*. De este modo los objetos no irán cambiando de lugar durante el transcurso de los casos de uso y sobre todo se pudo elegir la disposición exacta de cada uno de ellos. Una vez creado el *JPanel,* todos los elementos serán añadidos a este en lugar de al *JFrame.* Se destaca que todos los paneles implementados sobre el *panelBase* también tendrán seleccionado el *Layout* como nulo, por el mismo motivo anteriormente citado.

```
frame.setSize(1000,600);
frame.setLocationRelativeTo(null);
frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
frame.setVisible(true);

panelBase.setLayout(null);
panelBase.setBackground(new java.awt.Color(153, 153, 255));

frame.add(panelBase);
```

El tamaño elegido es acorde al tamaño reflejado en el requisito RF.18. Cuando se ejecute el *software* la vista se situará en el medio de la pantalla y se cerrará cuando se presione el botón rojo de cerrar.

Para la representación de las cartas representadas, como se explicó en el diseño en la sección 3.3.7, se ha optado por la implementación de *JButtons* rectangulares con un tamaño de 70x90 píxeles cuando están de cara y de dorso de 36x60 píxeles, como se observan estas últimas en la Figura 29. Pero esto generó un problema de superposición en las cartas, ya que si se pasaba el "cursor" del ratón por encima de cualquiera de ellas que estuviese detrás de otra, superponía la que tendría que estar tapada sobre la que tapa. Para evitar ese problema, las cartas que estén tapadas por otras, se implementan con un tamaño 70x40 para las cartas del jugador Sur y de 18x60 para los dorsos de las cartas de Sur y de Norte. Para las cartas de Este y Oeste, que están tapadas durante toda la partida hasta que son jugadas, tienen un tamaño de 36x12. De este modo, no hay unos *botones* detrás de otros, sino que son contiguos, pero representan

únicamente parte de la imagen. El mismo problema sucedía con las cartas del muerto, pero al tener otra disposición, las cartas que están tapadas son implementadas con un tamaño de 70x28 píxeles, como se observa en la Figura 41. Si una carta queda destapada, el programa cambia su tamaño a la dimensión de 70x96 píxeles.

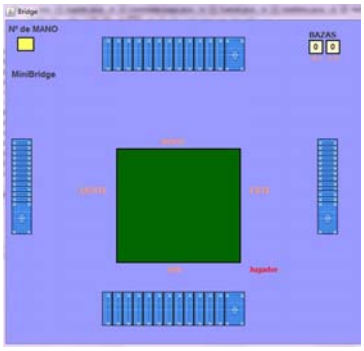


Figura 42. Disposición de los dorsos de las cartas

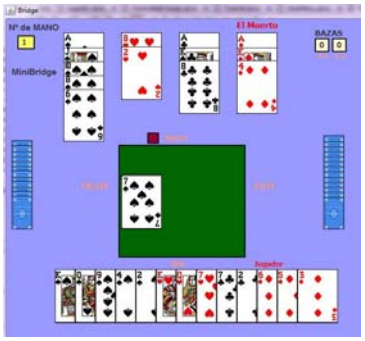


Figura 41. Disposición de las caras de las cartas

Esto ocurre exactamente igual con las *voces* que van dando los *jugadores* durante la fase de subasta. Se implementarán diferentes tamaños para los *botones* que representan las voces, en función de las dimensiones diseñadas en el apartado 3. Los *botones* que representan las voces tendrán un tamaño de 81x50 píxeles para voces con un nivel entre 1 y 3 ambos incluidos, de tamaño 70x50 píxeles para las voces de los niveles 4 y 5, un tamaño de 60x50 píxeles para las voces de los niveles 6 y 7 y las voces especiales un tamaño de 50x50 píxeles. Cuando estas *voces* son tapadas por otras posteriormente subastadas, se modifica el tamaño del *JButton* para que no ocurran problemas de superposición como ocurrió con las cartas de los jugadores. Por ello estas pasan a tener un tamaño de 30x50 píxeles. Cabe destacar que estos tamaños son para las voces de los jugadores Norte y Sur, para los jugadores Este y Oeste el tamaño es el mismo, pero intercambiando la coordenada *X* por la coordenada *Y*. Figura 43.

Figura 43. Disposición de las voces sobre la mesa

Para crear una interfaz agradable a la vista del usuario, se ha utilizado una herramienta *Kuler*, *de Adobe* (23). Esta herramienta permite crear gamas cromáticas complementarias y suplementarias. Los colores creados por esta herramienta a partir del color corporativo del cliente, RN.06, para representar los paneles más importantes, a excepción de la mesa de juego, son los siguientes:

- *panelBase* : (153, 153, 255)
- *panelBidding* : (178, 58,27)
- *textPane* : (191, 196, 125)

Para la mesa de juego se ha elegido el color verde apropiado para un tapete, concretamente el color (0, 102, 0).

Se ha tomado una imagen, Figura 44, de la herramienta *Kuler* generando los colores específicos para nuestros elementos a partir del color base.

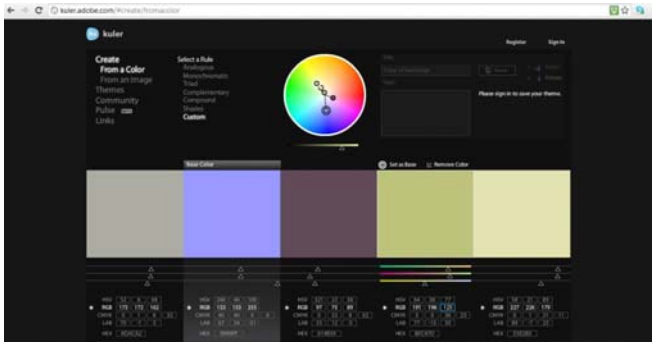


Figura 44. Herramienta *Kuler*

- **Bidding Box**

Para reflejar el *bidding box*, se ha implementado un *JButton* para cada una de las 38 voces de las que está compuesto. El tamaño es 40x27 píxeles, especificado en el diseño (sección 3.3.7). Estos *JButtons* se dispondrán por filas y en columnas y se añadirán al *panelBidding* de la clase *VistaPartida* como representa la Figura 45.

Figura 45. Disposición del *bidding box*

- **JLabel**

Todas las palabras y carteles que aparecen directamente sobre el *panelBase* son implementadas con un *JLabel*. Cada uno de ellos tendrá un tamaño y algunos estarán escritos en letra capital. El factor común de estos *JLabels* es el tipo de fuente con el que son implementados, *Tahoma* en color negro, a excepción de los *JLabels* que indican los nombres de los jugadores que serán naranjas sin brillo.

- **JButton.**

Tanto las cartas, como el *bidding box* y las opciones de juego están implementadas con un *ActionListener* para saber cuándo son pulsados, la clase *ActionEvent*.

```
m12.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
     m12ActionPerformed();
    }
});
 manoNueva.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        repartir();
    }
});
```

Cada *JButton* que representa las cartas de las manos de Norte y de Sur implementa un método *ActionPerformed()* para que *VistaPartida* y *ControladorJuego* analicen a qué carta está asociado.

```
private void m12ActionPerformed() {


    int posicion=this.sacarinosPosiciones[12];
         if(norteLibre==true){
            if(turno>0){
               Palo auxPalo=control.getPaloPrimero();
              int contador=this.cartasPaloEnNorteTotal(auxPalo);
                if(contador==0){
                    this.jugarCartaNorte(posicion,Palo.ST,s12);
                }
                if(contador>0){
                    this.jugarCartaNorte(posicion,auxPalo,s12);
                }
             }
            if (turno==0){

               this.jugarCartaNorte(posicion,Palo.ST,s12);
            }
            if(validacionNorte>0){
            norteLibre=false;
           }
        }
    }
```

Este método en concreto, representa el método que tiene lugar cada vez que se presiona el *botón m12*, es decir, la carta doce del Muerto. Esta clase

identifica qué *carta* ha sido escogida. Este método implementa el método *jugarCartaNorte().*

Cada vez que el usuario selecciona una voz del *bidding box* mediante un *botón*, sucede exactamente lo mismo que ocurre con los *botones* que representan las cartas, a excepción de que además se llama a un método que inhabilita las voces de menor nivel.

.

```
subasta4ST.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      subastarVozPerformed(3,0,subasta4ST.getIcon());
    }
});
subasta4P.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      subastarVozPerformed(3,1,subasta4P.getIcon());
    }
});
```

Para reflejar las imágenes que tiene que representar cada *JButton*, se utilizará la clase *ImageIcon* que tendrá como parámetro un *String* que localiza su correspondiente imagen o icono ya leído.

```
this.subasta1P.setIcon(new ImageIcon ("C:/Users/Tito/Documents/NetBeansProjects/Bridge/Imagenes/Esquemas/Picas/1
this.subasta2P.setIcon(new ImageIcon ("C:/Users/Tito/Documents/NetBeansProjects/Bridge/Imagenes/Esquemas/Picas/2
```

- **Panel mesa y turnos**

Este panel es el encargado de representar las *cartas* y *voces,* jugadas por los jugadores durante la partida, mediante *JButtons* que tiene ya incorporados. Como se ha explicado en la sección 3.3.7 del este capítulo, la mesa tendrá un tamaño mayor durante la fase de subasta que durante la fase de carteo. Se ha optado por un tamaño de las mesas de 220x200 píxeles para la fase de carteo y de 350x300 píxeles para la fase de subasta. El panel mesa únicamente tiene añadidos cuatro *JButtons* de 70x90 píxeles, uno a cada lado de la misma, que representan las cartas que han sido jugadas por la baza, Figura 46. Estos *botones*, no se crean nuevos sino que se les cambia la visibilidad según corresponda. Los *JButtons* que reflejarán las voces se irán creando y añadiendo según proceda, porque no se puede saber previamente cuántas voces se darán ni el tamaño de las mismas. Estos *botones* están sujetos a cambio de tamaño si posteriormente su *jugador* da otra *voz*.

Los turnos serán representados con cuatro *JButton* que irán modificando su visibilidad en función de si *el jugador* al que representan está en turno o no.

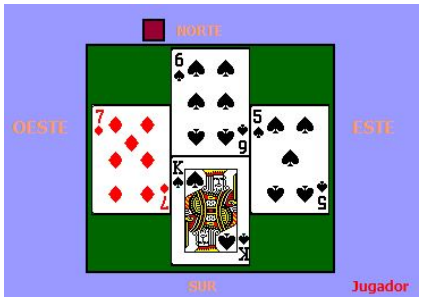


Figura 46. Disposición de las cartas jugadas sobre el panel mesa

- **JTextPane**

Para reflejar las ayudas del fichero de texto *(txt)* se utilizará un *JTextPane* con un *scroll*. Gracias al *scroll* podemos incorporar mayor cantidad de texto que el tamaño del *JTextPane* ya que podremos mover el *scroll* para ver las ayudas anteriores que no son reflejadas.

```
scroll.setViewportView(textPane);
scroll.setBounds(655,180, 300,370 );

panelBase.add(scroll);
```

Se implementan tres *JTextPane* más, uno para el número de mano, otro para el número de bazas de la pareja N-S y el último para el numero de bazas de la pareja E-O. Sus tamaños sean de 30x25 píxeles para el número de mano y de 25x25 píxeles para ambos marcadores de bazas, con fondo amarillo para resaltar el negro de la fuente. No se implementan con *scroll*.

El tipo técnico de fuente que representan los *JTextPane*s es *Arial 11 Bold*:

```
textPane.setFont(new java.awt.Font("Arial 11 Bold", 1, 13));
```

- **Panel de opciones.**

Para poder cumplir todos los requisitos, se ha implementado un panel de opciones para que el usuario pueda accionar las distintas posibilidades que se le facilitan durante el juego, representado en la Figura 47. Estos requisitos están detallados en la sección 2.7.2.

- RF.37- Empezar
- RF.39 -Deshacer
- RF.41 -Mano nueva
- RF.42 -Última Baza
- RF.43 -Repetir
- RF.40 - Salir
- RF.36 - Jugar Libre/Ayudas

Figura 47. Panel menú de juego

No todas las opciones caben en el *panel*, pero no todas se pueden implementar en todo momento, por ello los *botones* cambian de nombre y funcionalidad según vayan cambiando las fases del juego. Por ejemplo, una vez ha empezado el carteo, ya no se podrá volver a empezar por lo que ese *botón* el ser accionado representará la última baza, y no habrá última baza si no se ha empezado el carteo.

- **Vista general**

A continuación se detalla, en las Figuras 48 y 49, la visión general de la *VistaPartida* durante los casos de uso CU.02 y CU.03 de miniBridge y de Bridge respectivamente.

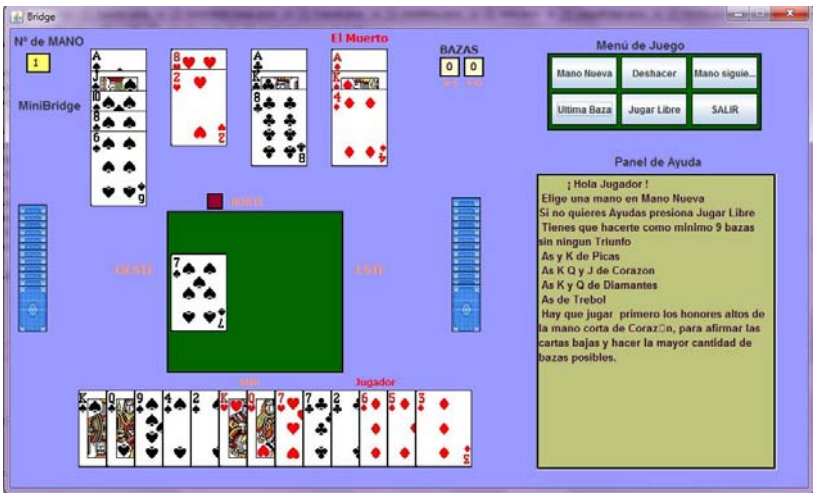


Figura 48. Vista general de la partida miniBridge

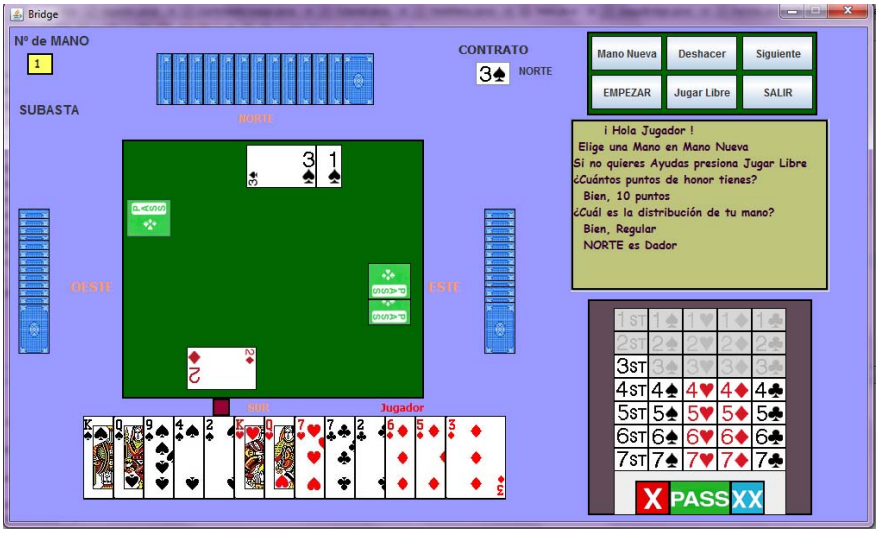


Figura 49. Vista general de la partida de Bridge

4.1.2 Subcomponente VistaTutorial

Para la creación de *VistaTutorial* tampoco se ha optado por utilización de herramienta para construir interfaces que facilita *NetBeans*.

- **Constructor**

  Esta clase tiene un único constructor cuyos parámetros son una variable que identifica qué tipo de tutorial tiene que representar y el nombre del usuario.

- **JFrame y JPanel**

  VistaTutorial también ha sido implementada con un *JFrame* y un *JPanel* con la misma disposición y objetivos que los especificados en la apartado 4.1.1 de este capítulo. El *Layout* de este panel también se implementará como nulo.

```
frame.setSize(1000,600);
frame.setLocationRelativeTo(null);
frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
frame.setVisible(true);

panelBase.setLayout(null);
panelBase.setBackground(new java.awt.Color(153, 153, 255));
frame.add(panelBase);
```

- **JLabel**

Esta vista está formada por un elemento principal que representará todas las imágenes de los tutoriales. Este elemento es un *JLabel* que tendrá el mismo tamaño que el *panelBase* en el que está contenido. Para cambiar las imágenes que representará el *JLabel* se ha implementado la clase *ImageIcon* como ocurre con los *JButtons*.

Para la implementación de las imágenes que representará el *JLabel* se ha utilizado la herramienta *Photoshop de Adobe CS3* (22), Figura 50. Esta herramienta facilita enormemente la implementación del diseño detallado en la sección 3.3.1. De este modo nos ahorramos la creación de muchos elementos en Java, por lo tanto, ahorro de tiempo de cara a la implementación de esta clase.

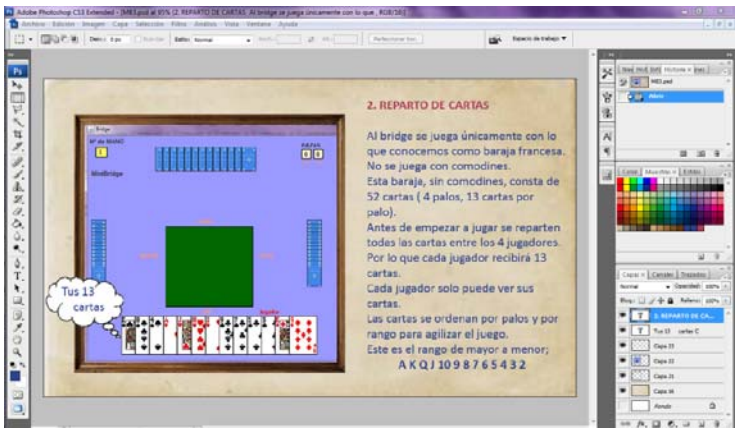


Figura 50. Photoshop creando los tutoriales

- **JButton**

Para cambiar las imágenes o poder volver a la Vista Menú se ha implementado en pantalla tres *JButtons* para accionar los métodos *siguiente()*, *anterior()* y *volver()*. Para ello, estos *botones* también estarán implementados con *ActionListener* para saber cuándo son ejecutados. Esto último se especifica en el apartado anterior en la parte de *JButton.*

- **Vista general**

A continuación se presenta la imagen general del subcomponente, Figura 51.

Figura 51. Vista general del tutorial

4.1.3 Subcomponente VistaMenu

Para la creación de *VistaMenu* tampoco se ha optado por utilización de herramienta para construir interfaces que facilita *NetBeans*.

- **Constructor**

  Esta clase posee un único constructor que tiene como único parámetro el nombre del usuario representándolo mediante un *JTexField* como se detalla posteriormente.

- **JFrame y JPanel**

  VistaMenu también ha sido implementada con un *JFrame* y un *JPanel* con la misma disposición y objetivos que los especificados en la apartado 4.1.1 y 4.1.2 de este capítulo. El *Layout* de todos los paneles de la vista se implementarán como nulos. Como excepción, el *panelBase* de esta clase tendrá el fondo azul, para darle más vistosidad al programa.

```
panelBase.setBackground(new Color(51,102,0));
```

  Se implementarán dos paneles diferentes para separar claramente los diferentes casos de uso. Estos paneles irán implementados con el mismo color que el *panelBase* de *VistaPartida.*

```
panelTutoriales.setBackground(new Color(153, 153, 255));
panelPartidas.setBackground(new Color(153, 153, 255));
```

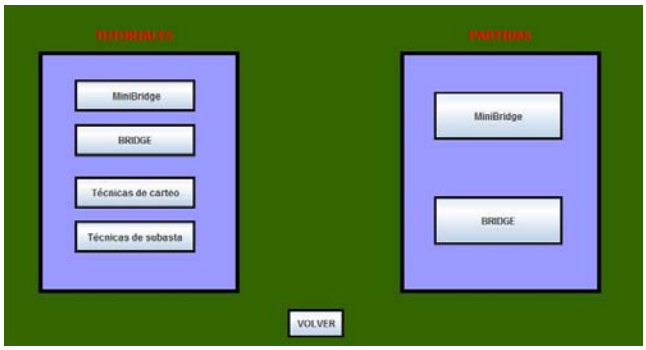


Figura 52. Paneles de casos de uso

- **JButton**

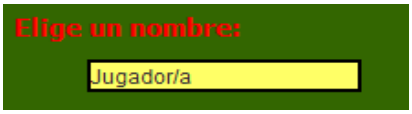
  Para poder acceder a los diferentes casos de uso, se implementan seis *botones* para acceder a los tres casos de uso especificados en la 2.6 del análisis. Estos *botones*, como se explicó anteriormente en la sección 4.2.1, serán implementados con un *ActionListener* para saber cuándo son ejecutados y así poder llamar al método correspondiente. También se incluirá sobre el panelBase un *JButton* que llame a la acción "volver", para implementar la *PantallaBienvenida*.

- **JTextField y JLabel**

A diferencia de las demás Vistas, esta implementará un *JTextField* para que el usuario pueda elegir el nombre y que la máquina lo recoja. El constructor de esta clase tiene como único parámetro el nombre del usuario, de este modo si el usuario ya incorporó un nombre y posteriormente seleccionó un caso de uso y decidió volver en un momento dado a *VistaMenu* no haría falta volver a incorporar dicho nombre porque ya se representa por defecto en el *JTextField*.

```
this.jTextField1= new JTextField();
this.jTextField1.setBounds( 90,90,150,20);
this.jTextField1.setBackground(new Color(255, 255, 102));
this.jTextField1.setBorder(BorderFactory.createLineBorder(new java.awt.Color(0, 0, 0), 2));
jTextField1.setText(nombreJugador);
```

Para la representación de los títulos se han implementado *JLabels* con fuente *Tahoma* de color rojo.



- **Vista general**

  A continuación se presenta la imagen general del subcomponente, Figura 53.

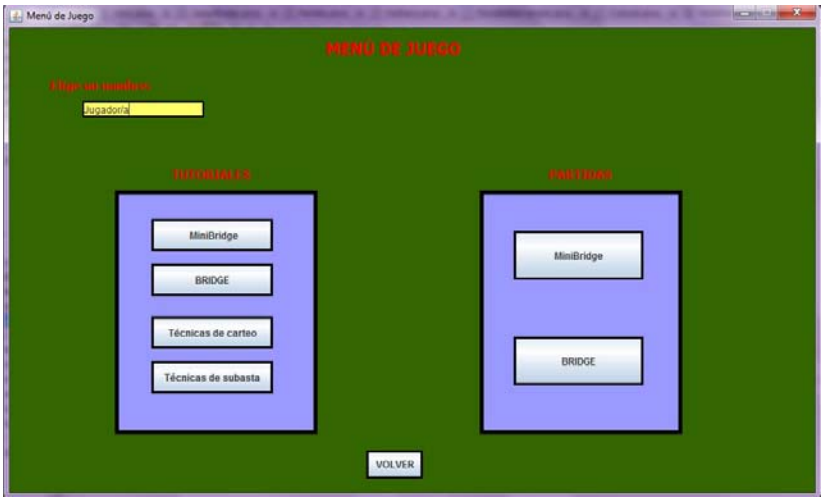


Figura 53. Vista general de VistaMenú

4.1.4 Clase PantallaBienvenida

Para esta la creación de esta clase sí se ha utilizado la herramienta *Palette* que proporciona *NetBeans,* debido a la simplicidad de la clase. La cual únicamente posee un *JLabel* que soportará la imagen y un *JButton*. Este *JButton*, al ser pulsado, implementará la clase *VistaMenu*. El funcionamiento de estos elementos se detalla en los apartados predecesores de este capítulo.

- **Vista general**

  A continuación se presenta la imagen general del subcomponente, Figura 54.

Figura 54. Vista general PantallaBienvenida.

4.2 Proceso de codificación del componente Controlador

Como se indica en el diseño, sección 3.2, el componente Controlador está formado por tres subcomponentes *ControladorJuego*, *JuegoBridge* y *Partida*. Además de detallar la implementación de estos componentes también se especificará la implementación de la clase *Jugador*.

4.2.1 Controlador Juego

Para cumplir sus cometidos, esta clase se diseñó con ciertos métodos, sección 3.3.4. A lo largo de este apartado se detalla la implementación de los más importantes.

- **Constructor**

  Como se detalló en el diseño, esta clase es implementada con dos constructores, uno para cada caso de uso, CU.02 y CU.03.

- **Métodos**

  La función principal de estos elementos está detallada en la sección 3.3.4.

  - **ejecutarSesion()**

```
public void ejecutarSesion(){

   if(subasta==true){

       this.iluminarTurnoVista(3);
       ejecutarSubasta();
    }

   if(subasta==false ){

        pantalla.imprimirImagenesNorte();
        ejecutarCarteo();

    }

}
```

  Cómo el usuario puede no saber qué fase tiene lugar a continuación en la mano. Por ello el método *ejecutarSesión()* se encarga de empezar la fase correspondiente en función de si la variable *boolean subasta* es falsa o verdadera. En un caso u otro llamará al método, *ejecutarSubasta()* o *ejecutarCarteo()*.

- **ejecutarCarteo()**

```
if(baza>1){

     vectorBaza=new Carta[4];
    if(turnos[0].getPosicion()==0  ){

        partida.getOeste().jugarCarta(turno, vectorBaza);

    }
    if(turnos[0].getPosicion()==1  ){

        pantalla.libreNorte();

    }
```

Este método no se ha implementado con una función *While* porque el carteo no es síncrono ya que el jugador no juega carta siempre en el mismo instante de tiempo. Por ello se ha implementado este método con llamadas a jugadores máquina y dejando libre la pantalla para el turno del usuario. Una vez el usuario juegue una *carta* se llamará de nuevo a jugar *carta* a otro *jugador*.

- **ejecutarSubasta()**

```
if(this.numeroVocesDadas==0){
    this.numeroVocesDadas++;

    dealer=partida.getDealer();
    if(dealer==0){

           pantalla.setTexPane(" OESTE es dador");

           partida.getOeste().darVozSubasta(desarrolloSubasta,


            }
```

Este método, al igual que el anterior, no es síncrono, por lo que tampoco se ha implementado con una función *While*. El *ControladorJuego* irá pidiendo *voces* a los distintos *jugadores* en turno. En caso del jugador usuario, será mediante el *bidding box*.

- **evaluacionGanadorBaza()**

```
public int evaluacionGanadorBaza(){

    Palo paloFirst = vectorBaza[0].getPalo();
             int mejor=turnos[0].getPosicion();
             int aux=0;
             int auxtriunfo=0;
             int contadorTriunfos=0;

       if(triunfa==Palo.ST){

             for(int j=0;j<=3;j++){
                   if (vectorBaza[j].getRango()>aux && vectorBaza[j].getPalo()==paloFirst ){

                              mejor=turnos[j].getPosicion();
                              aux=vectorBaza[j].getRango();
                   }
              }
         }

      else{
                 for(int m=0;m<=3;m++){
```

En función de las reglas de juego, este método recorre el *vectorBaza* para analizar cuál es *jugador* que ha jugado la carta que gana la baza. Para ello evalúa el *triunfo*, el *palo* con el que empezó la baza y los *rangos* de las *cartas* si coinciden con el *palo* inicial de la misma. El método devuelve un valor entero que indica la posición del *jugador* que ha ganado la baza.

- **asignacionTurnos()**

```
    public void asignacionTurnos(int mejor) {

     int t =mejor;
     int j=0;

        turnos= new Jugador[4];

     while (j<=3){

         turnos[j]=mesa[t];

       t=(t+1)%4;
       j++;

        }
```

Este método es implementado con el parámetro de la posición del *jugador* que ha ganado la baza. En función de ese parámetro crea los turnos para la siguiente baza gracias a un contador cíclico y otro lineal.

4.2.2 Juego Bridge y partida

Estas clases tienen un único constructor. Los métodos *generarBaraja()* y *generarBiddings()*, como se ha detallado anteriormente (sección 3.3.5), se encargan de generar todos los objetos necesarios para las partidas. Para ello lo único que harán será llamar, mediante funciones *For*, a los diferentes constructores de cada clase para generar dichos objetos.

```
public List<Carta> generarBaraja(){

        baraja = new LinkedList<Carta>();

        Carta carta = null;

        for(int i=2;i<=14;i++){
            for(int j=1;j<=4;j++){
                switch(j){

                    case 1:

                        carta = new Carta(i, Palo.CORAZONES,
```

```
public Voz[][] generarBiddings(){

        Voz[][] bidding= new Voz[7][5];
         Voz carton= null;

for (int i= 0; i<=6;i++){

   for (int j=0; j<=4; j++){

     switch(j){

            case 0:

                carton = new Voz((i+1), Palo.ST,0);
```

4.2.3 Jugador

Esta clase tiene un único constructor que tiene como parámetros un *nombre*, sus *cartas* y su *posición*. Pero lo principal de esta clase es cómo se ha implementado la lógica de juego representada parcialmente en la sección 2.10.

Para ello lo primero, cómo se explicó en el diseño, el jugador irá recorriendo sus cartas mediante sentencias *If* en función del turno que tiene en la baza, el palo de salida en la baza y el número de baza en la que se encuentra.

Por ello en primer lugar analizará si se encuentra "atacando" o "defendiendo".

```
if(turno>0){
paloPrimero=vectorBaza[0].getPalo();
}

if (triunfo==Palo.ST){
    if(turno>0){
      cartasDisponibles=depuracionAntiRenuncios();
      cartaAjugar = this.asistenciaST(this.paloPrimero);

    }
    if (turno==0){

        cartasDisponibles=cartas;

       cartaAjugar = this.defensaST();

    }
}
```

Una vez analizado la situación, llamará al método *asistencia()*, *defensa()*, o *ataque()*.

A su vez, estos métodos llamarán a diferentes métodos en función del análisis del juego. A continuación se enumeran los más importantes.

- *numeroCartasPalo()*
- *jugarLaCartaMasBaja()*
- *jugarLaCartaMasAlta()*
- *ganarConLaCartaMasBaja()*
- *comparadorDeVariedades()*
- *jugarPaloCompañero()*

Se destaca, que para poder llevar a cabo correctamente este proceso de selección el jugador no evalúa el atributo *rango* de las cartas, sino que evalúa el atributo *honor*, que irá cambiando a lo largo de la partida. Este atributo, cómo se explicó en la sección, va aumentando de valor según se vayan jugando las cartas que están por encima de la misma. Gracias a este atributo, se puede implementar un código mucho más sencillo y fiable.

4.3 Proceso de Codificación del componente Modelo

Este componente fue diseñado para leer e interpretar la información obtenida en los ficheros *CSV* de programa *Dealmaster Pro* y los ficheros de texto para las ayudas. Los ficheros están analizados en la secciones 2.4 y 2.5 y el diseño detallado del componente en la sección 3.3.6. Como se mencionó en tal sección, este componente está integrado dentro de la clase *JuegoBridge*.

Como se detalló en el diseño, este método estará compuesto por dos métodos, *leerFichero()* para los ficheros *CSV* y *leerAyudas()* para los ficheros de texto. Ambos

métodos fueron diseñados con las clases *FileReader* y *BufferedReader*.

- **leerFichero()**

```
public List<List<Carta>>  leerFichero (){

      try{
        FileReader file = new FileReader(ruta);
        BufferedReader buff = new BufferedReader(file);
        String fila=null;
        String paquete=null;
```

En primer lugar se crea un objeto *file*, de la clase *FileReader*, y se introduce en el constructor de la clase *BufferedReader*.

Una vez dentro del fichero, se procede a leer cada fila con el método *redLine()* de la clase *BufferedReader*. Una vez leída la fila se implementará la clase *StringTokenizer* y su método *nexToken()* para separar y seleccionar la fila en paquetes que como se analizó en la sección 2.4. En este fichero, los paquetes están separados por comas. Una vez separado en paquetes, se recorrerá con una función *For* la longitud del mismo, para así leer por separado cada uno de los caracteres que lo forman. Cada uno de los 52 caracteres iniciales es una carta.

```
 fila=buff.readLine();
 StringTokenizer miCelda = new StringTokenizer(fila, ",");

while(contadorCasillas<18){
        contadorCasillas++;

         if(contadorCasillas<=18){
        paquete= miCelda.nextToken().trim();

         }
            int longitud = 0;
            for(;longitud<paquete.length();longitud++){
                char letraActual = paquete.charAt(longitud);
                    rango=0;
                 if(contadorCasillas<=18){
                        switch(letraActual){
                            case 'A':
                                rango=14;
                               break;
                            case 'K':
                                rango=13;
                               break;
                            case 'Q':
                                rango=12;
                               break;
```

Una vez son leídos los caracteres, se asocian con las *cartas* de la baraja ya creadas y se asignan a sus respectivas *manos* en función de los parámetros analizados en la sección 2.4.

- **leerAyudas()**

El procedimiento llevado a cabo para la lectura e interpretación de las ayudas escritas en el fichero de texto, es el mismo que el mencionado anteriormente. La única excepción es que la ruta depende del número de mano que se esté jugando. Por ello este método escoge una ruta en la que influye el atributo *númeroMano* de la clase *JuegoBridge*.

La continuación para la interpretación de la subasta correcta y la obtención de caracteres es la misma que la del método anterior.

```
while((fila=buff.readLine())!=null){
  contadorCasillas++;
StringTokenizer miCelda = new StringTokenizer(fila, ";");

 paquete=miCelda.nextToken().trim();
 if(contadorCasillas==1){
     int longitud=0;

     for(;longitud<paquete.length();longitud++){
                char letraActual = paquete.charAt(longitud);

                    switch(letraActual){
                         case '7':
                                nivel=7;
                               break;
                           case '6':
                                nivel=6;
                               break;
```

4.4 Proceso de rediseño de las imágenes

Durante la implementación de los *JButtons* de la clase *VistaPartida* surgió un problema grave que dio lugar a un cambio en las imágenes de las cartas y de las voces.

Los *botones* que están tapados por otros, cuyo tamaño es inferior al tamaño real de las *cartas* y *voces*, no representaban bien las imágenes completas. Por ello se diseñaron unas imágenes nuevas en la que solo se representa una parte de la imagen inicial de las *cartas* y de las *voces.*

4.5 Resultados de las pruebas de aceptación

Una vez terminada la fase de implementación se procedió a realizar el plan de pruebas detallada en la sección 2.9 de este documento. En la siguiente tabla se detallan los resultados de las pruebas

Identificador	Resultado
PS.01	Prueba satisfactoria
PS.02	Prueba satisfactoria
PS.03	Prueba satisfactoria
PS.04	Prueba satisfactoria
PS.05	Prueba satisfactoria
PS.06	Prueba satisfactoria

Tabla 23. Resultado de las pruebas de software

Una vez todas las pruebas han dado un resultado satisfactorio se puede dar por concluido el *software*.

5. CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se incluyen las conclusiones alcanzadas tras la realización del proyecto. Al final del capítulo se detallan las posibles mejoras que se pueden aplicar en el *software* de aprendizaje para aumentar su funcionalidad.

5.1 Conclusiones finales

El proyecto realizado pretendía crear un *software* como herramienta de aprendizaje del juego de cartas Bridge, de una manera sencilla, atractiva e interactiva. Para ello se decidió llevar a cabo un proceso de ingeniería, en la cual se desarrollaría un sistema con los componentes necesarios para la creación de dicho *software* mediante el lenguaje de programación *Java*. Este sistema permite resolver los objetivos iniciales de modo que ahora existe una herramienta que permite familiarizarse con el juego y con las partidas sin la necesidad de conocimientos previos.

Por otro lado, comparando el *software* con los otros programas de Bridge, se considera que está a muy buen nivel tanto en el aspecto didáctico como en el técnico. Además este programa tiene unas características que no tiene ningún otro, dado que previamente se hizo un estudio de los programas más importantes de Bridge existentes. Cabe resaltar que la elección de *Java* como lenguaje para diseñar e implementar nuestro *software* ha sido un acierto.

La primera característica diferenciadora es que el idioma utilizado es el castellano. Otros programas de Bridge también tienen la posibilidad de seleccionar el castellano, pero estos programas son de niveles más avanzados. Otra de las características principales es el tutorial de iniciación con imágenes y ejemplos. Solo hay otro programa que ofrece tutoriales, pero éste requiere conocimientos previos de juego para su correcta comprensión, por lo que no es de iniciación. Los tutoriales que ofrece nuestro programa son sencillos y atractivos para el lector, motivando así su iniciación al juego. Otra característica principal de nuestro *software* es el sistema de ayudas que ofrece. Las ayudas son representadas durante la subasta y durante el carteo. Ayudas durante la subasta las ofrecen varios programas, pero ninguno durante la fase de carteo. Además, se recalca que las ayudas que ofrece nuestro programa son diferenciadoras, ya que le ayuda al usuario a evaluar correctamente la fuerza y distribución de su mano. La última característica de este *software* es que representa la subasta fielmente a la realidad, es decir, con las cartulinas del *bidding box* sobre la mesa al lado del jugador que la ha puesto. No hay otro programa de Bridge que tenga esta característica.

Estas características diferenciadoras dotan al programa de una capacidad única de aprendizaje y enseñanza, antes inexistente. Por lo que las expectativas de cumplir las motivaciones iniciales son esperanzadoras.

Además de estas características se menciona que gracias a la técnica de programación tradicional utilizada para la implementación de la lógica de juego de los jugadores máquina, se llega a la conclusión de dicha lógica de juego ha dado buen resultado, ya que estos jugadores cartean y subastan adecuadamente. Cabe destacar que no es uno de los mejores programas carteando, porque éste no era uno de los objetivos, pero tiene una idea de juego que permite al usuario entender la profundidad y dificultad de que entronca este maravilloso juego.

Como conclusión final se quiere añadir que el haber llevado a cabo un proceso de ingeniería para acometer el desarrollo de este proyecto, ha sido de gran ayuda y valor. El motivo fundamental para llegar a esta conclusión es que ha guiado extraordinariamente toda la organización del proyecto.

5.2 Dificultad del proyecto

A lo largo del proyecto se han presentado numerosos problemas y dificultades. A continuación se detallan las más importantes.

- **Tecnologías**

En primer lugar se destaca que las tecnologías empleadas para la realización del *software*, eran completamente desconocidas por el autor. En este aspecto existía un gran riesgo ya que no se podía predecir una duración aproximada para el desarrollo completo del proyecto ni su resultado final. El proceso de adaptación al lenguaje de programación *Java* y a la plataforma *NetBeans* fue duro y complicado, pero gracias a diversos libros y tutoriales específicos de Java y a las recomendaciones del tutor (24) se consiguió alcanzar un nivel de destreza suficiente para llevar a cabo el proyecto con éxito. Este proceso de adaptación y aprendizaje de *Java* no tuvo lugar únicamente durante la etapa inicial, sino que se desarrolló durante todo el proyecto ya a lo largo de las etapas de diseño e implementación se fueron aprendiendo nuevas clases y técnicas de programación en *Java*.

En cuanto al aspecto del análisis completo del Bridge, se destaca que no se llevó a cabo durante el proceso de elaboración del *software*. Esto es debido a que el autor del proyecto, antes de acometerlo, conocía todos los detalles sobre las reglas y objetivos del juego, así como las técnicas y métodos de enseñanza. A su vez, el autor sabía de antemano qué tipo de partidas implementar y sus características, cuán profunda debía ser la lógica de juego y las manos a incorporar. Gracias a esto se redujo considerablemente la duración del proyecto, porque es un juego complicado,

desconocido, con muchas alternativas de partidas, y que puede llegar a ser muy profundo.

- **Diseño e implementación de los componentes**.

Una vez superada la etapa de aprendizaje de las tecnologías, el mayor escollo fue la creación de la interfaz gráfica. En un primer lugar se optó por implementar los objetos en las vistas con la ayuda de la herramienta *Palette* que ofrece *NetBeans*. Estos elementos no se creaban y comportaban acorde a los requisitos del proyecto, por lo que se desestimó la utilización de dicha herramienta y se procedió modificar el diseño de las clases. Llegando a la conclusión de que la mejor solución, dada la inexperiencia del autor en este campo, era que todos los elementos se implementasen uno a uno, programando el propio autor todas sus características y poniendo el *Layout* de todos los paneles en modo *null.* Con esa nueva decisión se han obtenido unos resultados adecuados, cumpliendo todos los requisitos de la interfaz. Cabe destacar que los *Layouts* son muy útiles, pero dada la inexperiencia del autor, fue muy complicado su manejo y se evitó su uso.

Otro de los principales problemas fue la dificultad para conectar el componente Vista con el componente Controlador ya creado. Este problema fue debido a que en un primer momento se implementó la clase *ControladorJuego* con un *scanner*. La lógica de las partidas era llevada a cabo mediante *Systems.out.prints* y los datos introducidos en dicho *scanner*. Esta lógica estaba implementada con sentencias *While* para pedir las cartas a los jugadores de la partida. Con la interfaz, esta lógica daba problemas ya que se estaba implementando una secuencia síncrona cuando la elección de cartas por parte del usuario es asíncrona. Por este motivo se modificó el sistema de juego de *ControladorJuego,* quitando todos los *While* e implementado sentencias *If*.

El último problema a destacar fue la dificultad para incorporar la lógica de juego a los jugadores máquina. Inicialmente se implementó un sistema de sentencias *If* en cascada cuyo resultado fue realmente malo, ya que era muy complicado poner todas las situaciones y había veces que la máquina no elegía ninguna carta. Por ello se volvieron a diseñar los métodos para llegar a tal fin. En ese momento se llegó a una idea técnica que resolvió el problema notablemente y que a la vez simplificó enormemente el proceso de selección de cartas. Se decidió incorporar un atributo más a la clase *Carta*, concretamente el atributo "*honor*" como un *(int)*. Este atributo tiene como finalidad indicar el verdadero valor de la carta en ese preciso instante. Es decir, lo que en un primer momento puede ser un "7", si han sido jugadas todas las cartas superiores a esta, esta carta ahora tiene la misma fuerza que el As*.* Si por el contrario han sido jugadas todas las cartas superiores excepto una, este "7" tendrá el mismo valor que un rey *(K).* Este atributo es que el que manejarán los jugadores máquina, y el atributo *rango* de la *carta*, inmutable, lo utilizará *ControladorJuego* para evaluar los ganadores de la baza.

Con esta nueva última idea implementada, los jugadores máquina llevan cabo un buen carteo y una buena subasta.

- **Documentación**.

En un primer momento se pensó que el proceso de documentación sería mucho menos costoso de lo que realmente resultó ser. Esto fue debido al desconocimiento del autor sobre las técnicas, métodos, diagramas y lenguajes necesarios para la documentación de un proyecto de *software*. Con ello viene intrínseca la necesidad de aprender a manejar programas que permitan diseñar correctamente los diagramas de componentes, de clases y de secuencia. Cabe destacar que se cometieron varios errores y que los diagramas no eran del todo preciosos o correctos, por lo que hubo que repetirlos en varias ocasiones.

5.3 Líneas futuras

El *software* llevado a cabo, tiene varias líneas a seguir para su mejora. Estas líneas van referidas a diversos factores técnicos únicamente de cara a las partidas.

- **Memoria**

La principal desventaja de este programa es que no tiene ningún tipo de sistema de guardado para que cuando el usuario vuelva a ejecutar el programa no tenga que empezar otra vez desde principio. Una buena idea sería implementar un método de guardado de puntuaciones y manos. De este modo se puede mostrar al jugador la evolución técnica sobre las manos a lo largo del programa.

- **Partidas**

Otro factor principal a mejorar en las partidas, es la de otorgar al usuario la posibilidad de jugar manos como jugador defensor, incluyendo también varios niveles de juego, valorando también las vulnerabilidades y las voces de *Doblo* y *Redoblo*. Este tipo de partidas son de un nivel un poco más avanzado que el de iniciación, ya que en primer lugar se deben conocer correctamente los principales objetivos y características de este juego. Pero no olvidemos que nuestro programa resuelve este problema, por lo que se presenta como una buena idea.

Esta idea tiene una dificultad intrínseca por la que se decidió no ser llevada a cabo durante este proyecto. Y es que para ello se necesitaría una lógica de juego

muchísimo más avanzada por parte de los jugadores máquina. La inexperiencia del autor en el campo de la programación dificulta llegar a implementar correctamente este tipo de partida. Por este motivo, el autor, considera como una buena solución a este problema de programación lógica, el dotar de Inteligencia Artificial (I.A) al sistema porque no sería de gran utilidad un sistema de programación tradicional como el implementado en este programa.

- **Torneo no simultáneo**.

Si se llevase a cabo el desarrollo del punto anterior, sin importar exactamente qué tecnologías son utilizadas, se podría incorporar una modalidad de juego novedosa que no incluye ningún otro *software* de Bridge sin conexión a internet. Esta modalidad de juego es la creación del tipo de partida mediante un torneo no simultáneo. Con este tipo de partida, si el programa está siendo utilizado por un grupo de usuarios guiados por un mismo tutor, se conseguiría crear una competición entre ellos o el análisis de ejercicios creados por el autor, ya que el sistema permite incorporar manos a los profesionales de Bridge. Se destaca que para llevar a cabo esta idea se necesita un sistema de guardado como el anteriormente citado.

Este torneo no simultáneo consiste en que cada usuario, cuando buenamente pueda, juegue las manos que su profesor ha seleccionado. Al finalizar estas manos cada usuario obtendrá una puntuación en cada una de ellas. El tutor comparará las puntuaciones de todos los jugadores y obtendrá una clasificación final. Todos los usuarios han jugado las mismas manos y contra el mismo nivel de jugadores máquina.

BIBLIOGRAFÍA Y REFERENCIAS

1. **(IBC)**. *http://ibericaconsultinggroup.com/2012/06/21/aumento-en-la-demanda-de-servicios-de-fabrica-de-software-offshore-en-espana/*.

2. **Martínez García-Ciudad, José Miguel**. *Desarrollo de la inteligencia a través del Bridge*

3. **(AEB) Asociación Española de Bridge** http://www.aebridge.com/aebdynamic/dyncontent.asp?submenu=189&backkey=b

4. **(ACBL) American Contract Bridge League.** *http://www.acbl.org/learn/ltpb.html*

5. **(BBO) Bridge Base Online**. *http://www.bridgebase.com*

6. **Bridge Master 2000**. *http://bbi.bridgebase.com/software/bmdesc.html*

7. **Jack Bridge.** *http://www.jackbridge.com/eindex.htm*

8. **Blue Chip.** *http://www.bluechipbridge.co.uk/MiniBridge.htm*

9. **Bridge Baron.** *http://www.bridgebaron.com/*

10. **Q-Plus Bridge**. *http://www.q-plus.com/*

11. **Fun Bridge**. *http://www.funbridge.com/uk/index.asp*

12. **Micro Bridge**. *http://www.osk.3web.ne.jp/~mcbridge/*

13. **(GIB) Gilbert's Bridge.** *http://www.gibware.com/*

14. **Vélez Serrano, José F.** *Diseñar y programar, todo es empezar: Una introducción a la programación orientada a objetos usando UML y Java*.

15. **García Caballeira, Felix.** *Programación de lenguajes estructurados.*

16. **Flanagan,David**. *JavaScript: The definitive guide*.

17. **Groovy** *http://www.it.uc3m.es/spickin/docencia/comsoft/presentations/spanish/doc/Groovy.pdf*

18. **Dealmaster Pro.** *http://www.dealmaster.com/*

19. **Casos de Uso.** *http://www2.uah.es/jcaceres/capsulas/DiagramaCasosDeUso.pdf*

20. **(Metodología métrica vol.3)**. *http://alarcos.infcr.uclm.es/doc/ISOFTWAREI/Tema08.pdf*

21. **(UML). Lenguaje Unificado Modelado** *http://docs.kde.org/stable/es/kdesdk/umbrello/uml-basics.html*

22. ***Photoshop CS3 Adobe.*** *http://www.adobe.com/es/products/*

23. ***KulerAdobe.*** *http://kuler.adobe.com/*

24. **Eckel,Bruce.** *Piensa en Java 4º ed.*

    Sánchez Allende, Jesús. *Programación en Java.*

    Hunt, John. *Java for practitioners: Introduction and reference to Java and object orientation.*

25. **(MVC) Modelo Vista Controlador** *http://siul02.si.ehu.es/~alfredo/iso/06Patrones.pdf*

26. **API.** *http://docs.oracle.com/javase/6/docs/api/*

27. **Microsot Visio 2010.** *http://office.microsoft.com/es-es/visio/pagina-principal-de-visio-2010-FX010048786.aspx*

28. **NetBeans** *http://netbeans.org*

29. **Microsoft Office .** *http://www.microsoft.com/*

30. ***Dia 0.97.02.*** *http://projects.gnome.org/dia/*

31. **Google Chrome 5.0** *http://www.google.es/*

32. ***Club Squeeze Arte Bridge***. *Desarrollo de una partida de Bridge.*

33. **Amortizaciones**. *http://www.gesproem.es/2012/01/tablas-de-amortizacion-del-impuesto.html*

34. **Retenciones *IRPF*** *http://www.tucapital.es/calculadoras/retenciones-nomina-2012-2013/*

35. **Aportaciones Seguridad Social** *http://www.seg-social.es/Internet_1/Trabajadores/CotizacionRecaudaci10777/Basesytiposdecotiza36537/index.htm*

36. ***Calculadora Salarial*** *http://www.expansion.com/midinero/herramientas.html?vc=calculadora_salarios*

ANEXO I. GESTIÓN DEL PROYECTO

Este anexo incluye los aspectos del proceso de desarrollo del *software* relacionados con la planificación del proceso de ingeniería, presupuestos y medios técnicos empleados durante el mismo. En primer lugar se detalla el modelo del proceso elegido. Posteriormente se adjuntan las planificaciones iniciales y finales con diagramas de *Gantt* y los métodos técnicos empleados. Por último se adjunta el análisis completo de costes.

Modelo de proceso elegido

El modelo de proceso que se ha considerado más adecuado para desarrollo del *software* ha sido el proceso de ingeniería, detallado en la sección 1.1 de este documento. Este proceso se llevará a cabo con un modelo de proceso en cascada adaptado para poder retroceder a fases anteriores para realizar las modificaciones pertinentes y debido a la dificultad y desconocimiento de las tecnologías aplicadas para el diseño e implementación del proceso, Figura 55.

Las fases para el desarrollo de cada componente del proyecto son las siguientes:

- **Análisis**: identificación de los requisitos del componente, por medio de reuniones con el tutor del proyecto.

- **Diseño:** descomposición de cada componente en subcomponentes y el diseño del funcionamiento de cada uno. Elaboración de los diagramas de clases y de secuencia.

- **Implementación**: como su propio nombre indica, concierne la implementación de las clases de cada componente.

- **Pruebas**: proceso en el que se ejecutan las pruebas para la detección de errores en los componentes para poder ser corregidos.

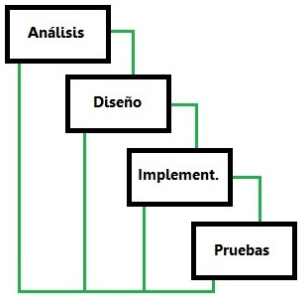


Figura 55. Modelo en cascada

Planificación del trabajo

A continuación se detalla la planificación estimada antes de acometer el proyecto y la posterior planificación que tuvo lugar. Así mismo, se incluyen las conclusiones de las desviaciones entre ambas planificaciones. El proyecto ha sido realizado por un único recurso personal, el autor del presente documento que ha realizado todo el proceso de ingeniería del *software* para el desarrollo del mismo, la asistencia a reuniones, revisiones, envíos de correos electrónicos, etc. Se destaca la labor del tutor, que actúa como cliente y consultor a todos los efectos durante todo el proceso.

El análisis de costes y de presupuestos se enmarca en el escenario de una empresa que es contratada por un cliente para la realización de este proyecto. Esta empresa tiene en plantilla un empleado con un salario anual fijo. El horario de trabajo del personal es de lunes a viernes es de 8:00*a.m* a 17:00*p.m* con una hora de descanso a las 13:00 horas. Los días festivos nacionales no computan como horario de trabajo. Además se incluyen las festividades del Jueves y Viernes Santos, así como el día de la Virgen y el dos de mayo.

Planificación inicial

En el momento del estudio de la planificación inicial se estimó que se podría desarrollar el componente Vista de una forma distinta a la solución final implementada. Se creía necesaria la utilización de la herramienta que facilita *NetBeans 6.9.1* para crear interfaces para disminuir tiempos durante el proceso de implementación de este componente.

La planificación inicial del proceso de ingeniería se representa con siguiente diagrama de *Gantt*:

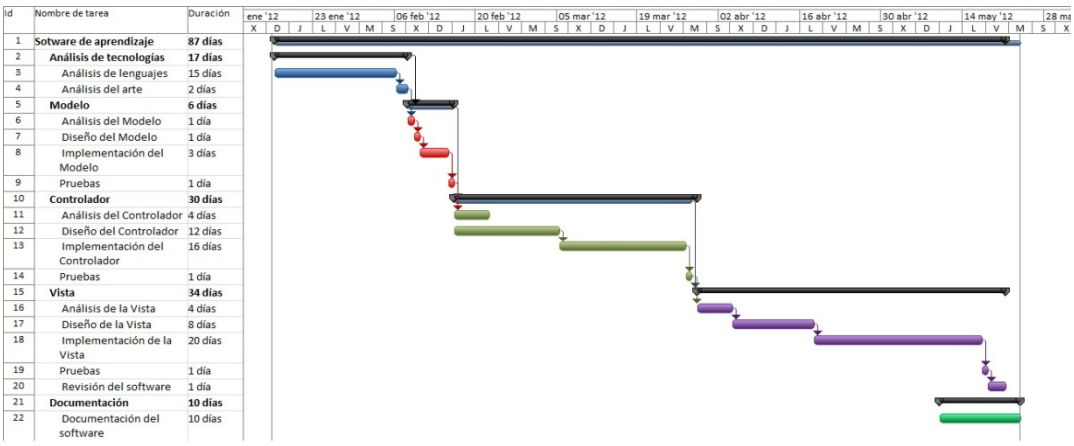
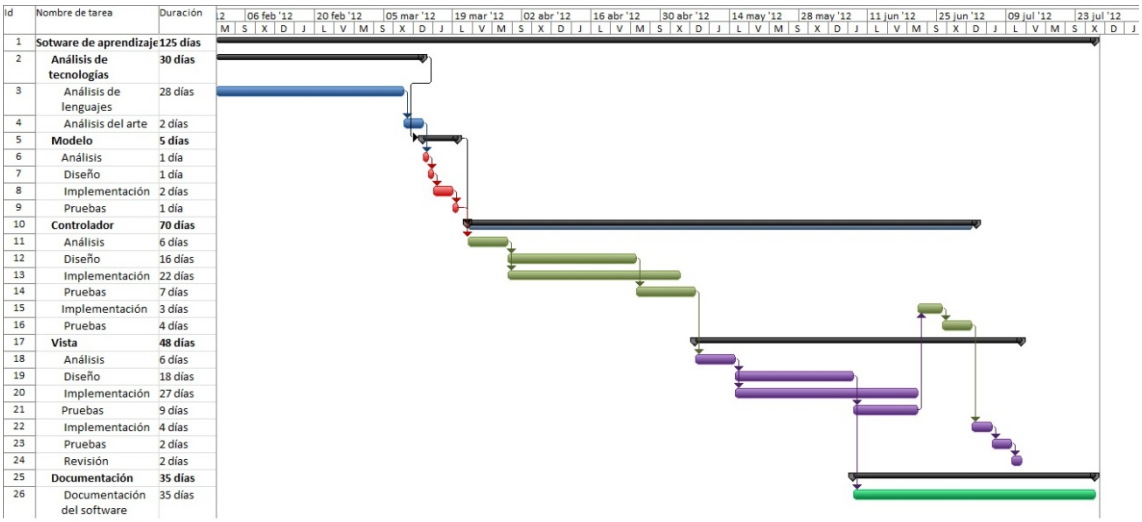


Figura 55. Diagrama de Gantt inicial

La estimación inicial predecía que el proyecto se acometería en 4 meses, de los cuales serían 87 días netos y 704 horas de trabajo.

Desarrollo real

El desarrollo real se recoge en el siguiente diagrama de *Gantt*:



Id	Nombre de tarea	Duración
1	**Sotware de aprendizaje**	**125 días**
2	**Análisis de tecnologías**	**30 días**
3	Análisis de lenguajes	28 días
4	Análisis del arte	2 días
5	**Modelo**	**5 días**
6	Análisis	1 día
7	Diseño	1 día
8	Implementación	2 días
9	Pruebas	1 día
10	**Controlador**	**70 días**
11	Análisis	6 días
12	Diseño	16 días
13	Implementación	22 días
14	Pruebas	7 días
15	Implementación	3 días
16	Pruebas	4 días
17	**Vista**	**48 días**
18	Análisis	6 días
19	Diseño	18 días
20	Implementación	27 días
21	Pruebas	9 días
22	Implementación	4 días
23	Pruebas	2 días
24	Revisión	2 días
25	**Documentación**	**35 días**
26	Documentación del software	35 días

Figura 56. Diagrama de Gantt final

Como se puede apreciar, inicialmente se estimó acometer el proyecto a mediados de enero y que estuviese finalizado a mediados de mayo. Finalmente, el proyecto empezó dos semanas más tarde y concluyó a finales de julio. La duración total fue de 6 meses de los cuales fueron 125 días netos y 1056 horas de trabajo.

Análisis de la desviación

Si se compara la estimación inicial con el resultado final, se observa que la duración del proyecto ha sido superada en un 33% respecto a la estimación inicial. Las diferencias frente a la planificación inicial se producen principalmente durante la etapa inicial de aprendizaje y el desarrollo del Controlador y la Vista. A continuación se especifican y analizan los motivos de tales diferencias.

- **Desconocimiento de la tecnología**

  El proceso de aprendizaje de la tecnología de programación Java ha resultado más complicado de lo que en un momento dado se estimó.

Esta dificultad redunda en las bases de programación del ingeniero autor de este proyecto. Estas bases son únicamente de programación *FORTRAN*, que es, a diferencia de Java, un lenguaje de programación de muy bajo nivel. Por lo que para un ingeniero técnico industrial con especialidad en mecánica no resulta sencillo el aprendizaje y manejo avanzado de este lenguaje de programación. En el marco de la formación como ingeniero, el aprendizaje y manejo de este lenguaje ha proporcionado al autor unos conocimientos muy útiles y un aumento notorio en la capacidad de análisis.

- **Solución técnica distinta**

  En la planificación inicial se pensó en la utilización de la herramienta *Palette,* de *NetBeans*, para la implementación arquitectural de todos los objetos de la Vista. A lo largo de la implementación estos elementos con atributos creados automáticamente por la *Palette* no cumplían los requisitos expuestos en el capítulo de análisis, sección 2.8. Estos elementos, en concreto los *botones* que representan las cartas, modificaban su disposición a lo largo de la pantalla según iban apareciendo diferentes eventos como, otras cartas, marcadores de turno o la desaparición de cartas ya jugadas. Además esta herramienta no permitía la disposición correcta de las cartas en la mesa, por los *Layout* y *GridLayout* predefinidos. Los *Layouts* son muy útiles y agilizan la implementación, pero que, dada la experiencia del autor y las particularidades de la interfaz, fue conveniente tomar otra decisión.

  Por ello, como se indica en la sección 4.1.1, este cambio de idea fue descartar la opción de la utilización de dicha herramienta. La idea final fue la implementación de todos los elementos uno a uno creados "a mano" por el programador en lugar de por la *Palette.* Por ello se tuvo que volver a diseñar e implementar el componente Vista. Para ello, como se especifica en la anterior sección citada, se anulan los *Layouts* para poder elegir la posición concreta de cada elemento de la Vista. Esto aumentó considerablemente los tiempos de implementación de los elementos de la Vista, pero permitió la correcta implementación de requisitos.

Medios técnicos Empleados

En este apartado se detallan todos los medios técnicos que ha sido necesarios para el desarrollo de este proyecto. Estos medios son los siguientes;

Hardware

El equipo utilizado para todas las tareas y procesos de este proyecto ha sido un *Acer Aspire 5742 Intel i5*, con 4 *gigabytes* de *RAM*, ejecutando el sistema operativo, de 64 *bits*, de *Microsoft Windows 7*.

Software

A continuación se detallan todos los *softwares* empleados para la realización de este proyecto:

- **Entorno de desarrollo**

  Se ha empleado el entorno *NetBeans 6.9.1* (28) para la implementación de todos los componentes y clases del *software*.

- **Navegador Web.**

  Se ha utilizado el navegador web *Google Crome 5.0* (31) a lo largo de todo el proyecto

- **Diseño de imágenes**

  Para el diseño de todas las imágenes se ha utilizado la herramienta Adobe Photoshop CS3 (22).

- **Elaboración de la documentación**

  La documentación del documento ha sido elaborada gracias a los siguientes programas:

  - El presente documento, que contiene la memoria, ha sido elaborado mediante *Microsoft Word 2007*, versión estudiantes (29).

  - La presentación del proyecto ha sido implementada y creada con el *software Microsoft Powerpoint 2007*, versión estudiantes (29).

  - El análisis económico ha sido analizado y llevado a cabo con el *software Microsoft Excel 2007*, versión estudiantes (29).

  - La planificación del proyecto ha sido planificada con el *software Microsoft Project 2007*, versión estudiantes (29).

- Los diagramas *UML* de componentes y de clases, han sido realizados y diseñados con el *software DIA v 0.97.02* (30).

- Los diagramas *UML* de secuencia han sido elaborados con el programa *Microsoft Visio 2010* (27).

Análisis económico del proyecto

En esta sección se incluirá todo el desglose de costes del proyecto, organizados por concepto. A su vez, al final de esta sección se hará un análisis de la desviación entre los costes estimados inicialmente y los reales.

Para ello se plantea una empresa que tiene en plantilla a un empleado al que le paga una nómina anual en doce plazos fijos. La empresa alquila una oficina en un edificio de oficinas mediante contratos de un año de duración prorrogables.

Análisis de costes

En este apartado se analizarán y detallarán los costes de empleados, directos e indirectos así como las amortizaciones de los activos del proyecto.

- **Costes de personal**

Salario y coste de la empresa del personal	Anual €	Mensual €
Coste total del empleado para la empresa	**37.303,66 €**	**3.108,64** €
Coste salario bruto empleado	28.500,00	2.375,00
Aportaciones Seguridad Social empresa	8.803,66	733,64
Salario bruto empleado	**28.500,00 €**	**2.375,00 €**
Aportación Seguridad Social empleado	1.339,50	111,63
Retenciones IRPF	5.130,00	427,50
Salario neto	**22.030,50 €**	**1.835,88** €

Tabla 24. Costes de personal

En esta tabla se desglosan los costes del empleado para la empresa y el salario neto anual (36) que el propietario recibe en doce pagas. Para el cálculo de las aportaciones a la seguridad social por parte de la empresa (35) y las aportaciones y retenciones por parte del empleado (34) , se ha obtenido información de las siguientes referencias.

- **Activos amortizables**

Estos activos son los medios técnicos utilizados para el desarrollo del proyecto, especificados en el apartado anterior de este anexo.

Activos amortizables		€	€ sin IVA*	Plazo de amortización (años)	Coeficiente de amortización	Amortización anual	Amortización mensual
Hardware	*Pc Portátil ACER Aspire 5742*	649,00	532,18	4,00	25,00%	133,05 €	11,09 €
Software	*Java*	-	-	-	-	- €	- €
	Netbeans 6.9.1	-	-	-	-	- €	- €
	API	-	-	-	-	- €	- €
	Google Chromme 5.0	-	-	-	-	- €	- €
	Microsoft Office 2010 (estudiantes)	130,00	106,60	3,03	33,00%	35,18 €	2,93 €
	Microsoft Visio 2010	-	-	-	-	- €	- €
	Adobe CS3	-	-	-	-	- €	- €
	DIA v 0.97.02.	-	-	-	-	- €	- €
Total amortización						**168,23 €**	**14,02 €**

Tabla 25. Activos amortizables

***** En el momento de la compra, antes del 1 de septiembre de 2012, el IVA era el 18%. En esta tabla de amortizaciones descontamos este impuesto porque la empresa lo puede deducir a fin de año. Para el cálculo de los años amortizables según el concepto y el coeficiente de amortización correspondiente, se ha usado la información de la tabla detallada en esta referencia (33).

- **Costes indirectos**

Costes indirectos	€ /mes
Alquiler de oficinas	900,00
Suministros (Gas, luz, electricidad, agua)	50,00
Tarifa plana internet	35,00
Teléfono móvil	24,00
Cuota Ingenieros sin fronteras (ISF)	20,00
***Total/mes** costes indirectos*	**1.029,00 €**

Tabla 26. Costes indirectos

El pago del alquiler de las oficinas es un pago anual, por eso no se computa como amortizable. Además se supone que únicamente se va a realizar este proyecto, por eso recae en el mismo todo el peso de todo el alquiler. Los suministros se han calculado en función de la media de los hogares madrileños. Las tarifas de teléfono e internet corresponden con los gastos mensuales del autor del proyecto en tales conceptos.

Anualmente se paga una cuota donativa por valor de 240,00 € a la ONG Ingenieros Sin Fronteras (ISF) como responsabilidad social corporativa.

- **Costes directos**

Concepto	€/mes
Coste personal (salario y aportaciones seguridad social)	3108,64
Otros costes incurridos (transportes, papelería, varios)	25,00
***Total/mes** costes directos*	**3.133,64**

S

Tabla 27. Costes directos

El coste de personal para la empresa es el calculado anteriormente. Los otros costes incurridos son una media del mismo concepto de proyectos anteriores.

- **Costes de contratación**

Concepto	€/h
Coste de personal por hora	35,00

Tabla 28. Costes de contratación

Para acometer este proyecto, el coste horario de trabajo del empleado es de 35 €/hora.

Costes iniciales

Según los resultados obtenidos en los diagramas de *Gantt* para la planificación inicial y el análisis de costes, se han deducido los diferentes costes y presupuestos que se detallan a continuación.

Presupuesto inicial del proyecto	
Duración estimada (meses)	4
Duración estimada (horas de trabajo)	704
Precio por hora de trabajo	35,00 €/h
Precio de venta del proyecto desarrollo de software (inicial/4 meses)	**24.640,00 €**

Tabla 29. Presupuesto inicial

Estimación de cuentas de resultados (Proyecto por 4 meses)	
Ingresos por el proyecto de desarrollo de software	**24.640,00** €
Total Costes directos	- 12.534,55 €
Coste personal (salario y aportaciones seguridad social)	- 12.434,55 €
Otros costes incurridos (papelería, varios)	- 100,00 €
Total costes indirectos	- 4.116,00 €
Alquiler de oficinas	- 3.600,00 €
Suministros (Gas, luz, electricidad, agua)	- 200,00 €
Tarifa plana internet	- 140,00 €
Teléfono móvil	- 96,00 €
Cuota Ingenieros sin fronteras (ISF)	- 80,00 €
Amortizaciones	- 56,07 €
Resultado final (margen)	**7.933,37 €**
Rentabilidad	**47,49%**

Tabla 30. Estimación inicial de cuentas

El cliente que contrata los servicios de la empresa, ha de abonar la cantidad fija de 24.640,00 € para acometer el desarrollo del proyecto. Como hemos sido muy laxos en la planificación y facturamos a 35€/h, el beneficio que obtendría la empresa sería de 7.933,37 €.

Costes finales

Como se especificó en el apartado de planificación de este anexo, el proyecto tuvo una duración real de 6 meses y 1056 horas de trabajo.

Presupuesto final Proyecto	
Duración real (meses)	6
Duración real (horas de trabajo)	1.056
Precio de venta del Proyecto de desarrollo de software (final/6 meses)	24.640,00€
Coste real por hora de trabajo facturada	23,33 €

Tabla 31. Presupuesto final

Cuenta de resultados final (Proyecto elaborado en 6 meses)	
Ingresos por el proyecto de desarrollo de software	**24.640,00** €
Total Costes directos	- 18.801,83 €
Coste personal (salarios, aportaciones seguridad social, retenciones IRPF)	- 18.651,83 €
Otros costes incurridos (papelería, varios)	- 150,00 €
Total costes indirectos	- 6.174,00 €
Alquiler de oficinas	- 5.400,00 €
Suministros (Gas, luz, electricidad, agua)	- 300,00 €
Tarifa plana internet	- 210,00 €
Teléfono móvil	- 144,00 €
Cuota Ingenieros sin fronteras (ISF)	- 120,00 €
Amortizaciones	- 84,11 €
Resultado final	**- 419,94 €**
Rentabilidad	**-1,68%**

Tabla 32. Cuenta final de resultados

Debido a la incorrecta estimación de la planificación inicial, la rentabilidad ha sido negativa, teniendo que abonar la empresa 419,94 € de su capital. Respecto del resultado inicial, la empresa ha perdido el beneficio de 7.933,37 € y la diferencia es de 8.353,31 € respecto al beneficio calculado inicialmente.

ANEXO II. MANUAL DE USUARIO

En este anexo se presenta un breve manual de usuario del *software* de aprendizaje desarrollado en este proyecto.

Pantalla inicial

Al iniciar el programa, este nos recibe con una pantalla que presenta el nombre del proyecto y los creadores como se observa en la siguiente figura. Si se pulsa sobre la tecla "EMPEZAR" pasaremos al menú de juego.

Figura 57. Pantalla inicial

Menú de juego

En esta pantalla se podrá elegir el nombre del usuario y seleccionar los tutoriales y partidas tanto de miniBridge como de Bridge. Si se pulsa sobre uno de los tutoriales, programa representará la pantalla con las imágenes correspondientes al tutorial elegido. Si por el contrario, se pulsa sobre una de las partidas, el programa representará la pantalla con todos los elementos necesarios para jugar el tipo de partida seleccionada. Si se teclea sobre el *botón* "volver", el programa volverá a la pantalla inicial.

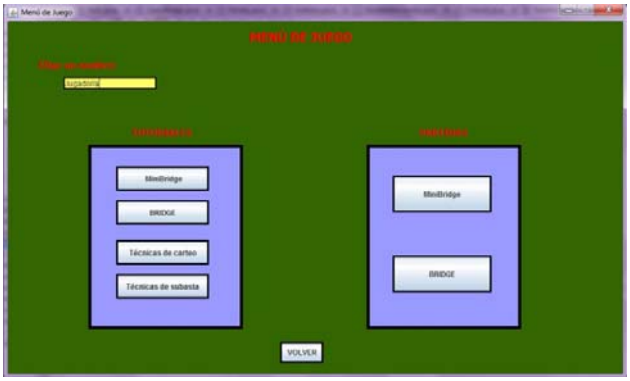


Figura 58. Menú de juego

Tutoriales

Los tutoriales serán representados con diferentes imágenes, como puede observarse en la Figura 59. Si se teclea sobre la flecha que señala a nuestra derecha, el programa representará la siguiente diapositiva, si por el contrario, se teclea la flecha que señala a nuestra izquierda, el programa representará de nuevo la diapositiva anterior. Si se teclea sobre el botón "Volver", el programa volverá a la pantalla con el menú de juego.

Figura 59.Tutoriales

Partida de miniBridge

Si se ha seleccionado en el menú la opción de partida de miniBridge, para repartir una mano se debe pulsar la tecla "Mano Nueva". Una vez repartida la mano, para proceder al carteo se pulsará el botón "Empezar". Si se desea jugar una carta basta con teclear encima de la misma. Una carta que sea pulsada y siga en la mano, significa que esa carta por cuestión de reglas o de turnos no puede jugarse.

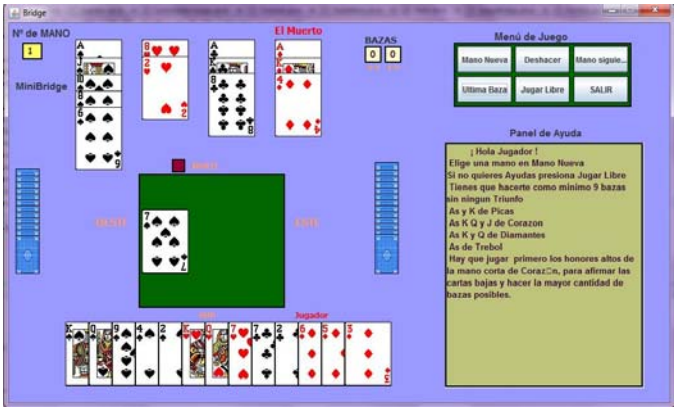


Figura 60. Partida de miniBridge

Una vez finalizada cada baza, para pasar a la siguiente, se podrá teclear encima de cualquiera de las cartas de la mano o en juego, sobre el fondo azul o sobre la mesa de color verde. Si se teclea sobre el botón "Última Baza", el programa representará la última baza jugada. Tecleando el botón "Deshacer", las cartas jugadas durante la baza volverán a las manos de sus propietarios.

En el caso de querer jugar sin ayudas habrá que teclear sobre el botón "Jugar Libre", si se quieren volver a activar las ayudas se tendrá que pulsar nuevamente sobre el mismo botón. Si se desea pasar a la siguiente mano o repetirla se tendrá que teclear los *botones* "Mano Nueva" o "Repetir". Si se quiere volver al menú de juego para seleccionar otra actividad, ha de pulsarse el botón " VOLVER" respectivamente.

Partida de Bridge

Si se ha seleccionado en el menú la opción de partida de miniBridge, para repartir una mano se debe pulsar la tecla "Mano Nueva". Una vez repartida la mano, para proceder a la subasta se pulsará el botón "Empezar". Si se desea subastar una voz basta con teclear encima del *bidding box*. Tecleando el botón "Deshacer", las voces jugadas durante una vuelta volverán al *bidding box*.

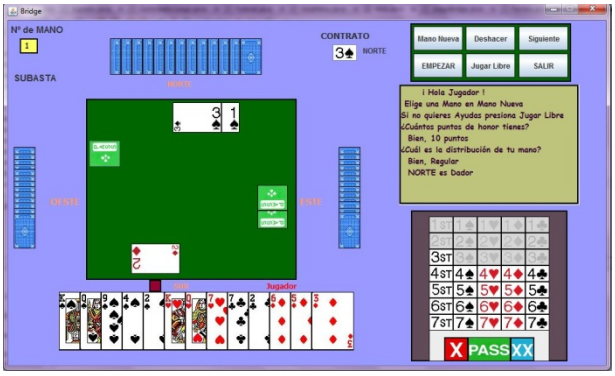


Figura 61. Partida de Bridge

En el caso de querer jugar sin ayudas habrá que teclear sobre el botón "Jugar Libre", si se quieren volver a activar las ayudas se tendrá que pulsar nuevamente sobre el mismo botón. Si se desea pasar a la siguiente mano o volver a jugar la misma se tendrá que teclear los botones "Mano Nueva" o "Repetir" respectivamente. En caso de querer volver al menú de juego, ha de pulsarse el botón "Volver"

Una vez termine la subasta la pantalla y sus funciones serán las mimas que en la partida de miniBridge explicadas en el apartado anterior.

ANEXO III. INTRODUCCIÓN A LA PARTIDA DE BRIDGE

Introducción y objetivos

En este anexo vamos a conocer los objetivos, reglas y pasos del Bridge para poder llevar a cabo las partidas correctamente (32). Aquí no veremos una lección de cómo jugar bien estratégicamente al Bridge, eso se detalla en el *software*. Aquí no está incluida la partida de miniBridge porque se incluyó en el *software* como método didáctico.

Este capítulo se centra en el sistema y reglas de este juego para ayudarte a comprenderlas. Exactamente lo mismo que si alguien enseña a mover las piezas del ajedrez. Una cosa es saber mover las fichas sin infringir las reglas y otra distinta es jugar con buena técnica.

A continuación se explican algunos términos para facilitar la comprensión del documento, además de los facilitados al inicio del documento (sección 1.4).

- **Representación de una carta**: ♥7 (primero el palo y luego el rango).

- **Representación de una voz**: 4♠ (primero el rango y luego el palo).

El objetivo principal del juego es hacer bazas, es decir, la pareja que más bazas consiga consigue una mayor puntuación si el contrato de la subasta ha sido adecuado.

Presentación de la baraja

Como ya sabemos, la baraja que se utiliza para jugar al Bridge es la **baraja inglesa**, vulgarmente conocida como baraja francesa, pero esta atribución es incorrecta. La baraja inglesa procede de la baraja francesa pero con distinta simbología. Se jugará con la baraja inglesa sin comodines. La baraja sin comodines consta de 52 cartas repartidas en cuatro palos.

- Picas ♠
- Corazón ♥
- Diamante ♦
- Trébol ♣

Cada palo lo forman **13 cartas** distintas en rango y valor. El rango, de mayor a menor, es el siguiente:

- A K Q J 10 9 8 7 6 5 4 3 2 -

Fases del Juego

A continuación se detallan las diferentes fases que tienen lugar en todas las partidas (respetando siempre el orden estricto de las fases).

1º Posición

Al Bridge se juega siempre por parejas. Se asignará una posición a cada pareja, que será la misma para toda la partida. Esta posición serán los puntos cardinales, hipotéticos, NORTE–SUR [N-S] para una pareja y ESTE-OESTE [E-O] para la otra. Los miembros de la pareja se disponen uno enfrente del otro, por eso esta distribución de los puntos cardinales. Por lo tanto un jugador será NORTE su compañero será SUR, otro será ESTE y su compañero OESTE.

Una mano de Bridge solo la puede jugar una pareja [N-S] y otra [E-O].

2º Reparto

Se reparte toda la baraja, las 52 cartas, entre los 4 jugadores. Cada jugador recibirá 13 cartas al inicio de la mano.

3º Juego

Una vez cada jugador haya recibido sus 13 cartas, la mano de Bridge consta de dos fases de juego estrictamente en este orden:

- 1º LA SUBASTA

- 2º EL CARTEO

-La Subasta.

Como ya se explicó anteriormente, el Bridge consiste en hacer bazas. En la fase de subasta se decide cuántas bazas promete conseguir como mínimo la pareja que más puje en número de bazas. Pero también se decidirá qué palo de los cuatro es el que triunfa sobre los demás o si no va a triunfar ninguno (ST - Sin Triunfo).

Para llevar a cabo la subasta es necesario un *bidding box* (caja de voces) explicado en detalle más abajo. Aquí no se van a jugar las cartas, eso será en el carteo. Pero las propias cartas influirán en las bazas que se irán prometiendo a lo largo de la subasta. En esta fase se subastarán diferentes voces del *bidding box.*

- El Carteo.

Como cada jugador recibe 13 cartas y cada uno ha de jugar carta cuando la jueguen los demás, en una mano de Bridge hay un total de 13 bazas.

Para llevar a cabo el carteo, como su propio nombre indica, sólo es necesario jugar las cartas para ir haciendo las bazas. Ya no es necesario el *bidding box.*

La pareja que no se ha quedado la subasta pasará a ser La Defensa en el carteo, es decir, tienen que intentar que la pareja que ha decidido la subasta (pareja declarante) no consiga hacerse las bazas que ha prometido o que se haga las mínimas posibles.

4º Puntuación.

Cada pareja obtendrá una puntuación en función de las bazas que haya conseguido durante el carteo respecto de las que se prometieron conseguir en la subasta y del palo del contrato.

Bidding Box

El *bidding box* (caja de voces) es un elemento necesario para la llevar a cabo la subasta. Está formado por cartulinas de plástico o papel grueso. Las cartulinas sirven para que los jugadores den las **voces.**

Figura 62. Bidding Box

El contenido del *bidding box* es el siguiente:

- **35** [7 filas x 5 columnas] cartulinas diferentes donde se indica el nivel de subasta en número de bazas y un dibujo que indica un palo o **ST** (Sin triunfo o **NT** dependiendo del modelo). Los niveles van del 1 al 7. Hay cinco cartulinas por cada nivel, los cuatro palos y el ST.

- **PASS** o **PASO** que como su propio nombre indica significa que no se tiene nada que decir.

- Las demás voces son más complicadas y se estudiarán en niveles más avanzados y no durante el proyecto.

Subasta

Me atrevería a decir que todo aprendiz de este maravilloso juego sabe el sistema que se lleva a cabo en cualquier subasta cotidiana. Pero por si alguien no lo sabe o hay que recordárselo, consiste en:

- El que más fuerte puja es el que se lleva el producto que se está subastando.

- Nadie puede pujar menos cantidad que la que se ha pujado en último lugar.

Con la subasta del Bridge sucede exactamente lo mismo:

- El que más fuerte puja en número de bazas es el que al final se queda con la subasta.

  Si posteriormente en el carteo consigue cumplir las bazas que ha prometido en la subasta, se le darán puntos, pero si no ha podido cumplir las bazas prometidas en la subasta, tendrá repercusiones (multas) que serán puntuaciones para la pareja defensora.

- Nadie puede pujar con una cantidad menor de bazas que la última cantidad de bazas pujada.

En la Subasta del Bridge, como en toda subasta, habrá todo tipo de "jugarretas", como por ejemplo:

- Los que pujen sin querer el producto para que los adversarios tengan que pujar más fuerte.

- Los que pujen aun sabiendo que no van a conseguir cumplir las bazas prometidas, pero con las multas (bazas de menos respecto de las prometidas)

obtienen menor repercusión que la recompensa de los adversarios por cumplir su puja en bazas que habían nombrado.

Estas jugadas son para un nivel de Bridge avanzado. Se citan ahora como explicación para la mejor comprensión de la subasta y para ver el alcance de las posibilidades del juego.

Sistema de la Subasta

En este apartado se detallan los procedimientos de la fase de subasta.

1º Comienzo

En las subastas que conocemos no hay turnos para pujar, pero en la subasta del Bridge sí que hay turnos. La subasta la comienza el jugador que sea el *dealer* **(dador)** en esa mano**.**

2º Continuación de la Subasta

El sistema de turnos es en sentido **horario**, es decir, el jugador tiene el turno cuando el jugador de su derecha acaba de dar una ***voz*** del *bidding box*.

3º Finalización

Cuando ha habido 3 voces de **PASS** consecutivas. Aquí se recogerán todas las voces, se meterán de nuevo en el *bidding box* y se procederá al **Carteo**.

Reglas de la subasta

- La asignación de *dealer* (o dador) a un jugador depende del tipo de partida (los tipos de partida se verán más adelante en este capítulo). Si es partida libre el *dealer* será el jugador que reparte la mano. Siempre se reparte en sentido horario de una en una, empezando por el adversario de la izquierda y dando previamente a cortar al oponente de la derecha. El siguiente en repartir será el jugador de su izquierda, siguiendo siempre el sentido horario. En partida de Bridge *duplicado* o por *equipos* vendrá indicado el *dealer* según el número de mano que se esté jugando.

- El sistema de turnos es horario. Nadie puede dar una voz hasta que el jugador de su derecha haya dicho alguna. No hay saltos de turno en ningún momento, por lo que el turno pasa inmediatamente al adversario de la izquierda.

- Cada jugador solo puede dar una única voz del *bidding box* por turno.

- La voz que se quiera dar tiene que superar la **fuerza*** de cualquier voz anteriormente dada por cualquier jugador, incluyendo las voces nuestro compañero, exceptuando la voz de **PASS** que se podrá dar siempre.

*** Fuerza de las voces**

La fuerza de una voz se mide primero por su **nivel**, y dentro de cada nivel por su rango de fuerza en función del palo. Es el mismo rango de palo en los siete niveles.

A mayor rango mayor fuerza dentro del nivel.

Orden de Fuerza

1. El nivel del 1 al 7. A mayor nivel mayor fuerza.
2. El Rango de fuerza de mayor a menor:

   ST ♠ ♥ ♦ ♣

3. La voz de **PASS** no tiene ninguna fuerza, ni rango ni nivel.

- La voz con más fuerza es el **7 ST y con menor fuerza de todas es 1♣**.

  Ejemplos:

  - La voz de 2 ♥ tiene mayor fuerza que la voz de 2♦

  - La voz de 4 ♣ tiene mayor fuerza que la voz de 3♠

  - La voz de **4 ST** tiene menor fuerza que la voz de 5♦

- Ningún jugador está obligado a subir el nivel o el rango de la subasta. Cualquier jugador puede dar la voz de nivel que quiera siempre y cuando supere en fuerza a la anterior, no estando obligado a dar la siguiente voz mayor en fuerza a la última anteriormente dada.

- La subasta finaliza cuando 3 jugadores dan la voz de **PASS** consecutivamente exceptuando la situación de que ningún jugador ha dado **voz de nivel** en la

primera vuelta, que en lugar de 3 **PASS** consecutivos son 4 **PASS** ya que cada jugador tiene el derecho al menos de dar una voz.

- Una vez ha terminado la subasta, ha habido un jugador que ha dado una voz de fuerza mayor que las demás. Esa pareja es la que después, en el carteo, ha de intentar cumplir como mínimo las bazas que ha prometido. Pasa a ser la pareja declarante y la otra pareja será la defensa.

- El número de bazas que tiene que hacer la pareja que se ha quedado con la subasta (pareja declarante) viene indicado en función del nivel en que haya terminado la subasta.

El número de bazas que tiene que hacer la pareja declarante es el número de nivel final de la subasta más seis adicionales por defecto.

Número de bazas mínimas a realizar para obtener puntos es:

Bazas mínimas = Nivel Final + 6

Ejemplo 1º; si la subasta ha terminado en el nivel **2**, la pareja declarante ha de ganar durante el Carteo, como mínimo, 8 bazas (2+6). Si lo consigue obtendrá puntos y por las bazas extra también. Pero por cada baza de menos respecto de las prometidas (multas) darán puntos a la pareja defensora.

Ejemplo 2º; si el nivel al que ha terminado la subasta es el **4** la pareja declarante ha de hacerse durante el carteo, como mínimo, 10 bazas (4+6).

Y si por ejemplo el nivel al que ha terminado la subasta ha sido el **7**, nivel máximo, dice que va a hacer 13 bazas (7+6), es decir, **TODAS** las bazas.

- Al finalizar la subasta no solo se ha decidido el número de bazas que se tiene que hacer la pareja Declarante sino que también se decide qué palo va a triunfar de los cuatro durante el carteo o si no va a triunfar ninguno (**ST**).

Toda voz de nivel lleva incorporado un palo o **ST.** Por lo que la última voz de nivel dada en último lugar determina qué palo o que ningún palo (**ST**) va a **Triunfar*** sobre los demás durante el carteo.

- La voz de nivel (el **PASS** no cuenta) dada en último lugar cuando ya ha finalizado la subasta, se llamará El Contrato. Como si fuese lo que se ha contratado.

Ejemplo 1º, si la última voz con nivel ha sido 3**ST** el contrato es 3**ST** (contrato a Sin Triunfo).

Ejemplo 2º si la última voz ha sido 2♦, ese será el contrato (contrato a Palo).

Solo puede haber un único palo de triunfo.

- Si todos han pasado, es decir, han dado la voz de **PASS** y no se ha dado ninguna voz de nivel, obviamente no habrá carteo, por lo que se pasa a jugar la siguiente mano. Ambas parejas recibirán cero puntos por esa mano.

- El Dealer puede decir en su primera voz el nivel de subasta que quiera o **PASS**.

- Cada voz subastada debe ser perfectamente visible por todos los jugadores y una vez puesta no se recogerá hasta la finalización de la subasta. Las siguientes voces que de cada jugador se deben colocar en la mesa de tal manera que no impida la visión de las anteriores voces dadas por los jugadores, así como ponerlas de tal manera que se vea que voces son anteriores y cuales posteriores.
  Esta regla no sigue un patrón universal ni exacto. No hay una manera exacta de poner unas voces respecto las otras, es suficiente con que las voces anteriores sean visibles y se entienda cuales se han dado primero, cuáles después y qué jugador las ha dado.

Figura 63. Subasta en una partida real

- En todo momento está prohibido que cualquier jugador muestre una/s carta/s a los demás.

- La Subasta siempre es anterior al Carteo.

- No se puede dar información al compañero por otro medio que no sea dando voces del *bidding box*.

Carteo

Como su propio nombre indica, en esta parte del juego se va a proceder a jugar las cartas para poder llevar a cabo las bazas.

Sistema del Carteo

1º Comienzo (La salida)

El jugador que comienza el carteo, el que tiene la salida, es el jugador de la pareja defensora a la izquierda del declarante.

El jugador declarante es el jugador de la pareja que se ha llevado el contrato en la subasta y que de ambos ha sido el primero de los jugadores de la pareja declarante en nombrar**,** durante la subasta, una voz con del mismo palo que el contrato final, o el primero de ambos que ha nombrado una voz de ST si el contrato es a Sin Triunfo. Por lo tanto la primera baza la inicia el jugador de la izquierda del jugador declarante.

2º Continuación del Carteo

El carteo también va regido por turnos **en sentido horario.** Inmediatamente después de que el jugador que inicia el carteo juega una carta, pasa el turno al jugador de su izquierda, que es el compañero del jugador declarante.

Aquí tiene lugar una de las peculiaridades por las que más se conoce el Bridge y uno de los factores que lo hace un gran juego mental. En este momento, cuando el compañero del declarante tiene el primer turno, éste jugador tiene que extender sobre la mesa a la vista de los demás jugadores sus 13 cartas ordenadas por palos. Estarán a la vista de todos durante el resto del carteo. Éste jugador y su mano pasan a ser **"El Muerto".**

Figura 64: El declarante, el muerto y la carta de salida

El Jugador Muerto no vuelve a intervenir en el resto de la mano. Sus cartas serán jugadas a elección de su compañero, el declarante. Una vez se han extendido las cartas del Muerto, el declarante elige la carta del muerto que crea conveniente

jugar, y pasará el turno al defensor de la izquierda del Muerto. El último en poner carta en la primera baza es el jugador declarante.

Una vez concluida cada baza empezará a jugar la siguiente baza el jugador ganador de la baza anterior. En las reglas del carteo se especifica perfectamente qué jugador de los cuatro es el que ha ganado cada baza.

El Muerto mantendrá su posición durante todo el carteo, solo que cuando le llegue el turno de baza, la carta a elegir será elegida por el declarante.

3º Finalización

El carteo concluye cuando ya se han realizado las 13 bazas, es decir, ya no quedan más cartas por jugar.

Reglas del Carteo

- El jugador que comienza el carteo, el que tiene la salida, es el jugador de la pareja defensora a la izquierda del declarante

- El orden de turnos para jugar las cartas en cada baza es horario.

- El Muerto siempre es el compañero del declarante. Desplegará sus cartas a los demás jugadores durante el resto del carteo inmediatamente después de que se haya producido "la salida".

- El declarante es el único que puede elegir qué cartas jugar del Muerto. En ningún momento el jugador Muerto ni nadie puede ayudar a elegir o decidir por el declarante qué carta jugar del Muerto en ninguna de las bazas.

- El jugador que inicia cada baza puede jugar cualquier carta que quiera de su mano. Si el Muerto es el que empieza la baza, la carta, como ya hemos dicho anteriormente, la elige única y exclusivamente el jugador declarante. Le toca jugar carta a su oponente de la izquierda.

- Cada baza solo la pueden componer cuatro cartas. Cada jugador solo puede jugar una carta por turno. Y una vez concluida la baza las cartas no volverán a ser jugadas en toda la partida, se quedaran en el lado de la mesa del jugador propietario.

➢ Asistir al Palo. Regla más importante del carteo.

Todos los jugadores, incluido el muerto, durante las 13 bazas están **obligados** a jugar una carta de su mano del palo correspondiente al palo de la carta con la que se ha iniciado esa misma baza, siempre y cuando tengan una o más cartas de ese palo en su mano. Esto es lo que se conoce como **Asistir al palo.** Esta regla influye en todos los palos. El incumplimiento de esta norma se llama Renuncio.

Si el jugador no tiene cartas del palo de inicio de esa baza, puede jugar la carta que quiera del palo que considere más oportuno.

Ejemplo1º, si el jugador que ha iniciado la baza juega el 5 de Corazón (♥5) todos los demás jugadores, incluido el Muerto, durante esa baza, están obligados a jugar cualquier carta del palo de corazón, mientras tengan cartas del este palo en su mano. Si no tienen cartas desde un principio del palo de corazón o se le han agotado en bazas anteriores puede jugar la carta que desee.

➢ ¿Qué jugador gana la baza?

Depende de si el Contrato es a Triunfo o a Sin Triunfo.

- **Contrato a Sin Triunfo (ST). No triunfa ningún palo.**

1º Un jugador, o el Muerto, gana la baza cuando él ha puesto la carta con mayor rango de fuerza del mismo palo que el palo de la primera carta jugada en esa baza.

El rango de fuerza dentro de los cuatro palos de mayor a menor es el siguiente:

- A K Q J 10 9 8 7 6 5 4 3 2 -

Aclaración: Un jugador no ganará una baza aunque el rango de fuerza de su carta sea mayor que la de los demás si el palo de su carta no corresponde con el palo de la carta con la que se ha iniciado esa baza.

Ejemplo:

Norte empieza la baza y le toca jugar al de su izquierda, que es ESTE.

Norte (muerto) sale del ♦**7**

Este juega el ♦**Q**

Sur juega el ♠**K** → **No tiene diamante y juega la carta que quiere.**

Oeste juega el ♦ **J**

Se deduce que la baza la gana **Este** con la ♦**Q,** aunque **Sur** haya puesto el ♠**K** porque el palo de la primera baza jugada es diamante y el único rango de fuerza que cuenta es el de diamante para esta baza. **Este** comenzaría la siguiente baza.

- **Contrato a PALO. El palo que Triunfa sobre los demás.**

1º Un jugador, o el Muerto, gana la baza cuando ha puesto la carta con mayor rango de fuerza del mismo palo que el palo de la primera carta jugada en esa baza**, exceptuando** que si en esa baza uno o más jugadores han jugado alguna carta del palo que triunfa. En caso de que se juegue una carta del palo de triunfo, ganará la baza el jugador que haya jugado la carta del palo de triunfo con mayor rango.

El Rango de Fuerza dentro de los cuatro palos, de mayor a menor, es el siguiente:

- A K Q J 10 9 8 7 6 5 4 3 2 -

Aclaración: Un jugador **no** ganará una baza aunque su carta sea la de mayor rango del palo de la primera carta jugada en la baza si, al menos, algún jugador ha jugado alguna carta del palo de triunfo por muy baja, pequeña, que sea esta última.

Ejemplos:

1º: Triunfo: ♥

Este empieza la baza y le toca jugar al de su izquierda, que es **Sur.**

Este sale del ♣**7**

Sur juega el ♣**A**

Oeste juega el ♥ **3** → **No tiene trébol y juega triunfo para ganar la baza.**

Norte juega el ♣ **4**

Como se observa, la baza la gana **Oeste** con ♥ **3** aunque **Sur** haya puesto el ♣ **A** porque el palo de triunfo es corazón y no hace falta que el **3** sea mayor que el **As** para ganar la baza.

Esto se conoce como **Fallar**; poner una carta de Triunfo para ganar una baza que no se ha iniciado con el palo de Triunfo.

Oeste comenzaría la siguiente baza.

2º: Triunfo: ♠

Norte empieza la baza y le toca jugar al de su izquierda, que es **Este**.

Norte sale del ♦**5**

Este juega el ♠**7** → **No tiene diamante y juega triunfo.**

Sur pone el ♠**9**→ **No tiene diamante y juega triunfo mayor para ganar la baza.**

Oeste juega el ♦**Q**

Como se puede observar la baza la gana **Sur con el ♠9 aunque Este** haya puesto el ♠**7,** siendo esta última del palo de triunfo. Porque gana la baza la carta del palo de triunfo más alta.

Esto se conoce como **sobrefallar**; ESTE ha fallado para intentar ganar la baza, pero SUR ha sobrefallado para llevarse la baza.

SUR comienza la siguiente baza.

- Si un jugador gana una baza, su compañero **también** la gana. Son una pareja, ambos la ganan. Pero la salida de la siguiente baza la llevara a cabo el jugador de la pareja que verdaderamente ha ganado la baza.

- Ningún jugador, ni el Muerto, está obligado ganar la baza si no quiere, es decir, no tiene que poner una carta de rango mayor que las anteriores. Pero siempre está obligado a jugar una carta del palo que ha comenzado la carta mientras tenga una o más cartas en ese palo **"Asistir al palo".**

- Ningún jugador, ni el muerto, están obligados a jugar una carta de triunfo si ya no tiene cartas del palo que ha comenzado la baza.

- Una vez se ha jugado una carta, ésta no vuelve a la mano sino que se queda bocabajo en la mesa en el lado del propio jugador. La carta se dispondrá de manera longitudinal al cuerpo del ese jugador si la baza ha sido ganadora y transversalmente al mismo si la baza ha sido perdedora. Esto se hace para facilitar el recuento final de bazas.

- Ningún jugador puede mostrar las cartas de la mano que no haya jugado, a excepción de Muerto. Tampoco ningún jugador puede voltear las cartas bocabajo de bazas ya jugadas para verlas, ni las propias, a excepción de las cuatro cartas de la última baza jugada, que las puede ver cualquiera de los tres jugadores cuando desee.

- Ningún jugador puede jugar carta hasta que no lo haya hecho su oponente (el muerto también es un oponente) de la derecha, exceptuando claramente al jugador que empieza la baza.

- El carteo acaba cuando se han jugado las 13 bazas, nunca antes.

Tipos de partidas

Además de las características de este juego, que lo hacen muy atractivo, divertido y saludable, existen diferentes tipos de partidas que hacen que este juego sea tan apasionante para tanta gente en el mundo entero, además de evaluarlo como deporte olímpico.

Al Bridge se juega por parejas pero se pueden jugar distintas modalidades, pero el sistema y reglas de juego son las mismas. Únicamente cambian las puntaciones.

Sabiendo jugar correctamente al Bridge a cualquier nivel puedes participar en cualquier tipo de partida lo único que cambia es el sistema de puntuación y el tipo de movimiento de las parejas. Esto último se explicara más adelante.

La modalidad más practicada es el Bridge Duplicado, en el cual, básicamente, las parejas van compitiendo contra las otras jugando exactamente la misma mano que ya han jugado anteriormente otras parejas. Cada pareja obtendrá la misma o diferente puntuación que las anteriores en función de lo bien que haya jugado. El ganador de esa mano (que no de la partida total) es el que consigue mayor puntuación.

Por este y otros muchos factores que se irán viendo, la puntuación no depende de la suerte que se tenga en recibir buenas cartas, sino en lo bien que se jueguen, ya que más parejas se habrán y se van a topar exactamente con la misma mano.

Partida MiniBridge

Esta modalidad es únicamente para cartear, es un Bridge sin subasta. Esta modalidad se utiliza exclusivamente como método académico.

Partida de Bridge Duplicado

Esta es la modalidad más atractiva para la mayoría de los jugadores de Bridge. Es la modalidad más popular para jugar torneos de Bridge y se necesitan al menos seis parejas para jugar. La principal característica de esta partida es que una vez se ha jugado una mano, ésta no se baraja sino que cada jugador guarda sus cartas en una tablilla (Figura 65) en el lugar de la posición cardinal del jugador. Esta tablilla pasa a la mesa siguiente. Con esto se pretende que exactamente la misma mano sea jugada por todas las parejas del torneo. Con este factor se reduce notablemente el factor suerte, ya que todos los jugadores recibirán las mismas cartas durante el torneo pero con esas mismas cartas obtendrán resultados y puntuaciones diferentes. Las parejas juegan contra todas las parejas y las tablillas que contienen las manos, se juegan en todas las mesas.

Figura 65. Tablilla para guardar las manos de una partida

Partida libre, Rubber

Cuando únicamente se dispone de dos parejas para jugar se juega una partida libre. En esta modalidad el factor suerte influye en un porcentaje suficientemente elevado como para no ser categorizado como deporte ya que las manos no las juega nadie más.

Partida por equipos

Esta es la modalidad más complicada de jugar y de entender. Esta modalidad es deporte olímpico por ser la que menor factor suerte tiene de todas.

Consiste en hacer equipos de parejas, mínimo dos parejas por equipo y tantos equipos como se quiera.

La característica principal es que en una mesa juegan dos parejas de dos equipos distintos y al mismo tiempo en otra u otras mesas juegan en las posiciones invertidas las parejas de estos dos equipos. Todos los equipos juegan contra todos los equipos.

Un ejemplo sencillo:

Equipo 1:

- Pareja 1: Iciar y Roberto
- Pareja 2: Inés y Antonio

Equipo 2:

Pareja 1: Almudena y Jorge
Pareja 2: Nela y Pedro

Una vez tenemos los equipos esta es la disposición de las parejas

Figura 66. Disposición de las parejas por equipos

El equipo 1 juega de N-S en la mesa 1 y de E-O en la mesa 2. Mientras que el equipo 2 juega de E-O en la mesa 1 y de N-S en la mesa 2. Las dos mesas

juegan exactamente la misma mano, obviamente con dos barajas distintas. Con esto se consigue que el resultado sea completamente equitativo con un leve factor de suerte. Al final de cada mano se comparan las puntuaciones de una y otra mesa.